НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.942::004.056.53

«До захисту допущено»
Завідувач кафедри
_____ І.А. Дичка
«___»_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Моделі та програмне забезпечення для виявлення SQL-ін'єкцій у веб-застосунках»

Виконав:
студент VI курсу, групи КП-71мп
Васін Євгеній Васильович _____
(підпис)

Науковий керівник:
Доцент кафедри програмного забезпечення
комп'ютерних систем ФПМ КПІ ім. Ігоря Сікорського,
к.т.н. Цуркан В.В. _____
(підпис)

Рецензент:
Начальник управління супроводження програми
інформатизації та нормативно-методичної діяльності
Департаменту інформатизації МВС України
к.т.н., доцент Дорогий Я.Ю. _____
(підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення» («Програмне забезпечення комп'ютерних та інформаційно-пошукових систем»)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Васіну Євгенію Васильовичу

1. Тема дисертації «Моделі та програмне забезпечення для виявлення SQL-ін'єкцій у веб-застосунках», науковий керівник дисертації Цуркан Василь Васильович, к.т.н., доцент, затверджені наказом по університету від «___»_____ 2018 р. №_____

2. Термін подання студентом дисертації «14» грудня 2018 р.

3. Об'єкт дослідження: процес виявлення SQL-ін'єкцій у веб-застосунках.

4. Предмет дослідження: програмне забезпечення для виявлення SQL- ін'єкцій.

5. Перелік завдань, які потрібно розробити:

- проаналізувати програмне забезпечення для виявлення SQL-ін'єкцій;
- побудувати концептуальну модель програмного забезпечення виявлення SQL-ін'єкцій;
- побудувати об'єктно-орієнтовану модель програмного забезпечення виявлення SQL-ін'єкцій;
- створити робочий проект програмного забезпечення для виявлення SQL-ін'єкцій;
- створити стартап проект програмного забезпечення для виявлення SQL-ін'єкцій.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

- діаграма варіантів використання програмного забезпечення;
- діаграма діяльності програмного забезпечення;
- архітектура програмного забезпечення виявлення SQL-ін’єкцій;
- діаграма компонентів програмного забезпечення.

7. Орієнтовний перелік публікацій:

- Програмний засіб виявляння SQL-ін’єкцій у веб-застосунках, XI наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК-2018-2).

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання «04» жовтня 2017 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.10.2017	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.12.2017	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	15.02.2018	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	05.04.2018	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.05.2018	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації	15.06.2018	

7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2018	05.11.2018	
8.	Оформлення текстової і графічної частини магістерської дисертації	04.12.2018	

Студент Є.В. Васін

Науковий керівник дисертації В.В. Цуркан

РЕФЕРАТ

Актуальність теми. Актуальність означеної теми обумовлена тим, що інформація являє собою певну цінність, має відповідне матеріальне або нематеріальне вираження та вимагає захисту від різноманітних за своєю сутністю несприятливих впливів. Одним із основних таких впливів є SQL-ін’єкція у веб-застосунках. Це обумовлено, наприклад, легкістю виявлення і експлуатування уразливостей баз даних і, як наслідок, їх використання зловмисниками. Тому, побудова моделей програмного забезпечення для виявлення SQL-ін’єкцій у веб-застосунках є актуальним завданням.

Об’єктом дослідження є процес виявлення SQL-ін’єкцій у веб-застосунках.

Предметом дослідження програмне забезпечення для виявлення SQL- ін’єкцій у веб-застосунках.

Мета роботи синтезувати програмне забезпечення для виявлення SQL-ін’єкцій у веб-застосунках.

Методи дослідження. Теоретичною основою дисертаційних досліджень є теорія моделювання процесів. Зокрема, теорія функціонального, процесного моделювання і моделювання потоків даних – для побудови концептуальної моделі програмного забезпечення виявлення SQL-ін’єкцій у веб-застосунках; теорія об’єктно-орієнтованого моделювання – для побудови об’єктно-орієнтованої моделі програмного забезпечення виявлення SQL-ін’єкцій у веб-застосунках.

Наукова новизна роботи полягає в одержанні таких результатів:

- уперше побудовано концептуальну модель програмного забезпечення виявлення SQL-ін’єкцій у веб-застосунках, використання якої дозволяє формалізування його роботу на рівні функцій, процесів і потоків даних, а також сформувати та обґрунтувати варіанти використання програмного забезпечення;

– уперше побудовано об’єктно-орієнтовану модель програмного забезпечення на основі формалізування його роботи на рівні функцій, процесів і потоків даних, використання якої дозволяє надати нову якість програмному забезпеченню при створенні або вдосконаленні, зокрема, забезпечити їх функціональну придатність до виявлення SQL-ін’єкцій у веб-застосунках.

Практична цінність отриманих результатів полягає у доведенні їх до практичного реалізування, а саме:

– розроблення програмного забезпечення виявлення SQL-ін’єкцій у веб-застосунках за його об’єктно-орієнтованою моделлю;
– виявлення SQL-ін’єкцій у веб-застосунках завдяки використанню розробленого програмного забезпечення.

Апробація роботи. Результати роботи апробовано на XI науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК-2018-2).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п’яти розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, оцінено сучасне програмне забезпечення виявлення SQL-ін’єкцій, обґрунтовано актуальність обраного напрямку досліджень, сформульовано мету і завдання дослідження.

У першому розділі проаналізовано програмне забезпечення виявлення SQL-ін’єкцій стосовно своєчасності оповіщення про них. За результатами такого аналізування показано необхідність створення відповідного програмного забезпечення, сформовано функціональні та не функціональні вимоги для його синтезування.

У другому розділі побудовано концептуальну модель програмного забезпечення виявлення SQL-ін’єкцій на функціональному, процесному рівнях, рівні потоків даних у графічних нотаціях IDEF0, IDEF3 та DFD. Це дозволило на функціональному рівні формалізувати його роботу набором

функцій. Тоді як на процесному рівні та рівні потоків даних описати взаємозв'язки між етапами оповіщення про SQL-ін'єкцію та обмін даних між ними.

У третьому розділі на основі концептуальної моделі побудовано об'єктно орієнтовану модель програмного забезпечення виявлення SQL-ін'єкцій у графічній нотації UML. Це дозволило визначити його варіанти використання, логічну та фізичну структури та, як наслідок, синтезувати програмне забезпечення виявлення SQL-ін'єкцій у веб-застосунках.

У четвертому розділі визначено характеристики, метрики якості та проведено функціональне тестування програмного забезпечення виявлення SQL-ін'єкцій у веб-застосунках. Наведено контрольні приклади тестування основних його варіантів використання. Визначено системні та апаратні вимоги до клієнтської та серверної частини програмного забезпечення. Описано кроки успішного встановлення і наведено інструкцію адміністратору для користування програмним забезпеченням виявлення SQL-ін'єкцій у веб-застосунках.

У п'ятому розділі визначено ринкові перспективи стартап-проекту програмного забезпечення виявлення SQL-ін'єкцій у веб-застосунках, графік та принципи організації виробництва, фінансовий аналіз та аналіз ризиків і заходи з просування пропозиції для інвесторів. Узагальнено етапи розроблення та виведення стартап-проекту на ринок.

У висновках проаналізовані отримані результати.

У додатках наведено структуру програмного забезпечення.

Магістерська дисертація виконана на 135 аркушах, містить 2 додатки та посилання на список використаних літературних джерел зі 50 найменувань. У роботі наведено 19 рисунків та 31 таблиця.

Ключові слова: SQL-ін'єкція, виявлення SQL-ін'єкцій, HTTP-запит, веб-застосунок, шаблони пошуку, концептуальна модель програмного забезпечення, об'єктно-орієнтована модель програмного забезпечення, програмне забезпечення виявлення SQL-ін'єкцій.

ABSTRACT

Relevance of the topic. The relevance of the topic is due to the fact that information is a certain value, has the appropriate material or intangible expression and requires protection from diverse in nature adverse effects. One of the key influences is SQL injection in web applications. This is due, for example, to the easy detection and exploitation of database vulnerabilities and, consequently, their use by malicious people. Therefore, the construction of software models for detecting SQL injection in web applications is an urgent task.

Object of research is the process of detecting SQL injection in the web application.

Subject of research software for detecting SQL injection in web applications.

Research objective to programmatically implement SQL Injection Detection Tool in Web Applications by building its object-oriented model.

Research methods. The theoretical basis of dissertation research is the theory of process modeling. In particular, the theory of functional, process modeling and data flow modeling - for building a conceptual model of software for detecting SQL injection in web applications; the theory of object-oriented modeling - for building an object-oriented software model for detecting SQL injection in web applications.

Scientific novelty work is to obtain such results:

- For the first time, a conceptual model of SQL Injection Detection Software in web applications was built, the use of which allows for formalizing its work at the level of functions, processes and data flows, as well as to form and justify the use of software software;
- For the first time, an object-oriented software model was built on the basis of formalizing its work at the level of functions, processes and data streams, the use of which allows providing a new quality software when creating or improving, in particular, to ensure their

functional suitability for the detection of SQL injections. in web applications.

Practical value of the results obtained is to bring them to practical realization, namely:

- developing software for detecting SQL injection in web applications for its object-oriented model;
- detecting SQL Injections in Web Applications through the use of developed software.

Approbation. The results of the work were tested at the XIth Scientific Conference of Graduate Students and PhD Students «Applied Mathematics and Computer» (PMK-2018-2).

Structure and content of the thesis. The master's dissertation consists of an introduction, five sections, conclusions and appendices.

The introduction provides a general description of the work, evaluated the modern software for the detection of SQL injections, justified the relevance of the chosen direction of research, formulated the purpose and objectives of the study.

The first section analyzes SQL Injection Detection Software for timely notification of them. According to the results of this analysis, the necessity of creating the corresponding software has been demonstrated, functional and non-functional requirements for its synthesis have been formed.

In the second section, a conceptual model for SQL Injection Detection Software is constructed at the functional, process levels, data streams in graphical notifications IDEF0, IDEF3 and DFD. This allowed the functional level to formalize its work by a set of functions. Then, at the process level and the data stream levels, describe the interrelationships between the stages of SQL injection alert and the exchange of data between them.

In the third section, based on the conceptual model, an object-oriented SQL injection model software in the graphical UML notation is built. This made it possible to determine its usage patterns, logical and physical

structure, and, consequently, to synthesize the SQL injection detection software in web applications.

The fourth section defines characteristics, quality metrics, and performs functional testing of SQL injection detection software in web applications. Examples of testing their main uses are given. The system and hardware requirements for the client and server part of the software are determined. Describes the steps for a successful installation and provides an instruction to the administrator to use the SQL Injection Detection Software in Web Applications.

The fifth section defines the market prospects for a software startup software for SQL Injection in web applications, a schedule and principles for organizing production, financial analysis and risk analysis, and measures to promote offers to investors. The stages of designing and launching a startup project on the market are summarized.

The conclusions are analyzed by the obtained results.

The annexes show the structure of the software.

The master's thesis is made on 135 sheets, contains 2 appendices and a link to the list of used literary sources from 50 titles. There are 19 drawings and 31 tables in the work.

Keywords: SQL Injection, SQL Injection Detection, HTTP Request, web application, search patterns, conceptual software model, object-oriented software model, SQL injection detection software.

РЕФЕРАТ

Актуальность темы. Актуальность обозначенной темы обусловлена тем, что информация представляет собой определенную ценность, имеет соответствующее материальное или нематериальное выражение и требует защиты от различных по своей сути неблагоприятных воздействий. Одним из основных таких воздействий является SQL-инъекция в веб-приложениях. Это обусловлено, например, легкостью обнаружения и эксплуатации уязвимостей баз данных и, как следствие, их использование злоумышленниками. Поэтому, построение моделей программного обеспечения для обнаружения SQL-инъекций в веб-приложениях является актуальной задачей.

Объєктом исследования является процесс выявления SQL-инъекций в веб-приложениях.

Предметом исследования является программное обеспечение для обнаружения SQL- инъекций в веб-приложениях.

Цель работы синтезировать программное обеспечение для обнаружения SQL-инъекций в веб-приложениях благодаря построению его моделей.

Методы исследования. Теоретической основой диссертационных исследований является теория моделирования процессов. В частности, теория функционального, процессного моделирования и моделирования потоков данных - для построения концептуальной модели программного обеспечения обнаружения SQL-инъекций в веб-приложениях; теория объектно-ориентированного моделирования - для построения объектно-ориентированной модели программного обеспечения обнаружения SQL-инъекций в веб-приложениях.

Научная новизна работы заключается в получении таких результатов:

– впервые построено концептуальную модель программного обеспечения обнаружения SQL-инъекций в веб-приложениях,

использование которой позволяет формализации его работу на уровне функций, процессов и потоков данных, а также сформировать и обосновать варианты использования программного обеспечения;

– впервые построено объектно-ориентированную модель программного обеспечения на основе формализации его работы на уровне функций, процессов и потоков данных, использование которой позволяет предоставить новое качество программному обеспечению при создании или совершенствовании, в частности, обеспечить их функциональную пригодность к выявлению SQL-инъекций в веб-приложениях.

Практическая ценность полученных результатов заключается в доведении их до практического реализуемость, а именно:

– разработка программного обеспечения обнаружения SQL-инъекций в веб-приложениях по его объектно-ориентированной моделью;
– выявление SQL-инъекций в веб-приложениях благодаря использованию разработанного программного обеспечения.

Апробация работи. Результаты работы прошли апробацию на XI научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» (ПМК-2018-2).

Структура та объем работы. Магистерская диссертация состоит из введения, пяти глав, заключения и приложений.

Во введении дано общая характеристика работы, оценены современное программное обеспечение обнаружения SQL-инъекций, обоснована актуальность выбранного направления исследований, сформулированы цели и задачи исследования.

В первой главе проанализированы программное обеспечение обнаружения SQL-инъекций относительно своевременности оповещения о них. По результатам такого анализа показана необходимость создания

соответствующего программного обеспечения, сформирован функциональные и нефункциональные требования для его синтезирования.

Во втором разделе построено концептуальную модель программного обеспечения обнаружения SQL-инъекций в функциональном, процессном уровне, уровне потоков данных в графической нотации IDEF0, IDEF3 и DFD. Это позволило на функциональном уровне формализовать его работу набором функций. Тогда как на процессном уровне и уровне потоков данных описать взаимосвязи между этапами оповещения о SQL-инъекцию и обмен данных между ними.

В третьем разделе на основе концептуальной модели построены объектно ориентированную модель программного обеспечения обнаружения SQL-инъекций в графической нотации UML. Это позволило определить его варианты использования, логическую и физическую структуры и, как следствие, синтезировать программное обеспечение обнаружения SQL-инъекций в веб-приложениях.

В четвертом разделе определены характеристики, метрики качества и проведено функциональное тестирование программного обеспечения обнаружения SQL-инъекций в веб-приложениях. Приведены контрольные примеры тестирование основных его вариантов использования. Определены системные и аппаратные требования к клиентской и серверной части программного обеспечения. Описаны шаги успешной установки и приведена инструкция администратору для пользования программным обеспечением обнаружения SQL-инъекций в веб-приложениях.

В пятом разделе определены рыночные перспективы стартап-проекта программного обеспечения обнаружения SQL-инъекций в веб-приложениях, график и принципы организации производства, финансовый анализ и анализ рисков и меры по продвижению предложения для инвесторов. Обзор этапы разработки и вывода стартап-проекта на рынок.

В выводах проанализированы полученные результаты.

В приложениях приведена структура программного обеспечения.

Магистерская диссертация выполнена на 135 листах, содержит 2 приложения и ссылки на список использованных литературных источников из 50 наименований. В работе приведены 19 рисунков и 31 таблиц.

Ключевые слова: SQL-инъекция, выявления SQL-инъекций, HTTP-запрос, веб-приложение, шаблоны поиска, концептуальная модель программного обеспечения, объектно-ориентированная модель программного обеспечения, программное обеспечение обнаружения SQL-инъекций.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

SQL (Structured Query Language) – є спеціальною мовою, що використовується в програмуванні, і призначена для управління даними, що зберігаються в реляційній системі управління базами даних, або для обробки потоку в системі управління потоком реляційних даних.

SQL Injection Attacks (SQLIAs) – це техніка введення коду, яка використовується для атаки програм, керованих даними, в яких некоректні оператори мови SQL вставляються в поле введення для виконання, що спричиняє сбій у веб-застосунку, а також доступ до бази даних зловмиснику.

Фазер – система, яка побудована на техніці тестування програмного забезпечення, часто автоматичній або напівавтоматичній, в основі якої лежить передача застосунку на вхід неправильних, несподіваних або випадкових даних [1].

LDAP – являє собою відкритий, нейтральний для постачальників, галузевий стандартний протокол застосування для доступу та обслуговування інформаційних служб розподіленого каталогу через мережу Інтернет-протоколу (IP) [2].

NoSQL – забезпечує механізм зберігання та вилучення даних, який моделюється засобами, відмінними від табличних зв’язків, що використовуються в реляційних базах даних [3].

XPath – це мова запиту для вибору вузлів з документа XML. Крім того, XPath може використовуватися для обчислення значень із вмісту XML-документа [4].

Rootuid – ідентифікатор користувача у UNIX системах.

XSS – це тип вразливості комп'ютерної безпеки, зазвичай зустрічається в веб-програмах. XSS дозволяє зловмисникам вставляти сценарії на стороні клієнта на веб-сторінки, які переглядають інші користувачі [5].

Directory Traversal Attack – перехрещення каталогів (або перетинання шляху) полягає у використанні недостатньої перевірки безпеки

користувацьких імен вхідних файлів, наприклад, що символи, що представляють «перехід до батьківського каталогу», передаються в API-файли [6].

ВСТУП

Важливим атрибутом нашого часу є глобальна інформаційна інтеграція, заснована на побудові комп'ютерних мереж масштабу підприємства та їх об'єднання за допомогою Інтернету.

Складність логічної та фізичної організації сучасних мереж призводить до об'єктивним труднощам при вирішенні питань управління та захисту мереж. У процесі експлуатації комп'ютерних мереж адміністраторам доводиться вирішувати дві головні задачі:

– діагностувати роботу мережі та підключених до неї серверів, робочих станцій і відповідного програмного забезпечення;
– захищати інформаційні ресурси мережі від несанкціонованої діяльності хакерів, впливу вірусів, мережевих хробаків і т. д., тобто забезпечувати їх конфіденційність, цілісність та доступність.

При вирішенні задач, зв'язаних з діагностикою та захистом мережевих ресурсів, центральним питанням є оперативне виявлення стану мережі або застосунку, які призводять до втрати повної чи часткової її працездатності, знищенню, спотворення або витоку інформації, що є наслідком відмов, збоїв випадкового характеру або результатом отримання зловмисником несанкціонованого доступу до мережевих ресурсів і веб-застосунків зокрема. Раннє виявлення таких впливів дозволить своєчасно усунути їх причину, а також запобігти нанесенню збитків.

Одним із основних таких впливів є SQL-ін'єкція у веб-застосунках. Це обумовлено, наприклад, легкістю виявлення і експлуатування уразливостей баз даних і, як наслідок, їх використання зловмисниками. Тому, побудова моделей програмного забезпечення для виявлення SQL-ін'єкцій у веб-застосунках є актуальним завданням.

1. ПРОБЛЕМАТИКА ПРОЕКТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ ВИЯВЛЕННЯ SQL-ІН'ЄКЦІЙ У ВЕБ-ЗАСТОСУНКАХ

1.1.Аналізування процесу виявлення SQL-ін'єкцій у веб-застосунках

Ін'єкція, або впровадження коду – це експлуатація комп'ютерної помилки, яка виникає при обробці недійсних даних. Засіб ін'єкції використовується зловмисником для введення коду в вразливу комп'ютерну програму та зміну курсу виконання. Результат успішної ін'єкції коду може бути катастрофічним, наприклад, завдяки поширенню комп'ютерних хробаків.

Вразливості введення коду (дефекти ін'єкцій) виникають, коли програма надсилає некоректні дані інтерпретатору. Дефекти ін'єкцій найчастіше зустрічаються в запитах SQL, LDAP, XPath або NoSQL; Команди ОС; XML-аналізатори, заголовки SMTP, аргументи програми тощо. Помилки прив'язки, як правило, простіше виявити при вивченні вихідного коду, ніж через тестування. Сканери та фазери можуть допомогти знайти недоліки ін'єкцій [7].

Ін'єкція може спричинити втрату даних, відсутність підзвітності або відмову у доступі. Ін'єкція іноді може призвести до повного захоплення хоста.

Деякі типи введення коду - це помилки в інтерпретації, що надають спеціальне значення лише користувачеві. Подібні помилки інтерпретації існують поза світом інформатики. У розпорядженні неможливо відрізнити власні імена від звичайних слів. Аналогічно, в деяких типах введення коду відбувається неможливість відрізнити введення користувача від системних команд.

Технології введення коду є популярними в системі зломів для отримання інформації, ескалації привілеїв або несанкціонованого доступу до системи.

Код ін'єкції може бути використаний зловмисно для багатьох цілей, у тому числі:

- швидка зміна значення в базі даних за допомогою SQL-ін’єкцій. Вплив цього може варіюватися від відхилення веб-застосунків до серйозного компромісу конфіденційних даних;
- встановлювання шкідливого програмного забезпечення або виконання зловмисного коду на сервері, вставляючи серверний сценарійовий код (наприклад, PHP або ASP);
- привілейована ескалація до кореневих прав за рахунок використання вразливостей Shell-ін’єкцій у rootuid-двійковій системі на UNIX або локальній системі, використовуючи службу в ОС Windows;
- напад на веб-користувачів за допомогою HTML/Script Injection (міжсторінкові сценарії).

У 2013 році 5,66% всіх вразливостей, що в цьому році були класифіковані як код ін'єкції, найвищий рік за рахунком. У 2015 році це зменшилося до 0,77%.

Код ін'єкції може бути використаний з гарними намірами; наприклад, зміна або налаштування поведінки програми або системи за допомогою введення коду може «примусити» систему до певної роботи, без будь-якого зловмисного наміру. Введення коду може, наприклад:

- введення нового стовпця, який не відображався в оригінальному дизайні сторінки результатів пошуку;
- запропонувати новий спосіб фільтрування, замовляти або групувати дані за допомогою поля, яке не відображається в стандартних функціях оригінального дизайну;
- додавання спеціальних частин, які можуть використовуватися для підключення до онлайнових ресурсів у позамережевій програмі.

Деякі користувачі можуть без вагань виконувати впровадження коду, оскільки вхідні дані, які вони надають програмі, не вважаються тими, які очікуються. Наприклад:

– те, що користувач може вважати правильним вводом, може містити токенові символи або символьні рядки, які розробник зарезервував для спеціального значення (можливо, «&» в «Shannon & Jason», або лапки, як у «Bub Slugger" McCracken»);
– користувач може подати неправильно сформований файл як вхід, який обробляється в одній програмі, але некоректний для приймаючої системи.

Ще одним вразливим вживанням коду може бути виявлення самих некоректних вад, з наміром виправити ці недоліки. Це відоме як тест на проникнення «білого капелюха».

Щоб уникнути проблем із кодуванням, використовують безпечну обробку вводу та виведення, наприклад:

– API такі, якщо вони правильно використовуються, захищені від усіх вхідних символів. Параметризовані запити (також відомі як "Скомпоновані запити", "пов'язані змінні") дозволяють інтерпретувати дані користувача поза рядком. Крім того, критерії API та подібні API відходять від концепції командних рядків, які будуть створені та інтерпретовані;
– виконання розподілу мов через систему статичного типу;
– верифікація вводу, така як білий лист лише відомих значень, це можна зробити на стороні клієнта за допомогою JavaScript, наприклад, або це можна зробити на стороні сервера, яка є більш безпечною;
– кодування вводу, наприклад уникаючи небезпечних символів. Наприклад, у PHP використовується функція htmlspecialchars (), щоб уникати спеціальних символів для безпечного виведення тексту в HTML, а mysqli :: real_escape_string () для виділення

даних, які будуть включені в SQL-запит, для захисту від ін'єкцій SQL;

- вихідне кодування, тобто запобігання нападам на HTML-ін'єкції (XSS) на відвідувачів веб-сайту;
- HttpOnly – це прапор для HTTP-файлів cookie, який, коли його встановлюється, не дозволяє взаємодії клієнтського сценарію з файлами cookie, тим самим запобігаючи певним атакам XSS [8];
- модульна розбивка корпусу від ядра;
- за допомогою SQL-ін'єкції можна використовувати параметризовані запити, збережені процедури, перевірку вхідних даних в білий список та інші, щоб допомогти пом'якшити проблеми з ін'єкцією коду [9].

Веб-застосунки часто вразливі до атак, які дають можливість зловмисникам легко отримати доступ до бази даних програми. Атака SQL-ін'єкцій відбувається, коли зловмисний користувач через спеціально створене поле спричиняє створення і надсилання запиту до веб-застосунку, який функціонує інакше, ніж це було задумано програмістом. Тому є доцільним розглянути даний тип ін’єкції коду.

Атака SQL-ін’єкції (SQL Injection Attacks, SQLIAs) відома як одна з найбільш поширених загроз безпеці керованих базами програм. Отже, недостатньо гарантії конфіденційності та цілісності цієї інформації. SQLIA - це клас атак на введення коду, який використовує відсутність верифікації користувача. Насправді, злочинці можуть формувати свій незаконний ввід як частини кінцевого рядка запиту, який працює в базах даних. Фінансові веб-додатки та секретні інформаційні системи можуть стати жертвами цієї вразливості, оскільки зловмисники, зловживаючи цією вразливістю, можуть загрожувати їх авторитету, цілісності та конфіденційності. Отже, розробники вирішували деякі захисні методи кодування, щоб усунути цю вразливість, але їх недостатньо.

Для запобігання SQLIA, оборонне кодування було запропоновано як рішення, але це дуже важко. Не тільки розробники намагаються поставити деякі елементи керування у вихідний код, але і атаки продовжують давати нові способи обійти ці елементи керування. З огляду на останню та найкращу практику оборонного кодування важко дотримуватися розробників. З іншого боку, впровадження найкращої практики оборонного кодування дуже складне і потребує спеціальних навичок. Ці проблеми мотивують потребу у вирішенні проблеми ін'єкції SQL.

SQLIA – це технологія хакерства, до якої зловмисник додає SQL-запити через поля вводу веб-додатків або приховані параметри для доступу до ресурсів. Відсутність перевірки вхідних даних у веб-додатках спричиняє стовідсоткову успішність атаки. У наступних прикладах припустимо, що веб-додаток отримує HTTP-запит від клієнта як вхідний і генерує SQL-випис як вихід для сервера баз даних на зворотному кінці. Наприклад, адміністратор буде автентифіковано після введення: employee id = 112 і пароль = admin. На рис. 1.1 описується логін з боку шкідливого користувача, що експлуатує вразливість SQL-ін’єкції. В основному вона складається з трьох етапів [10]:

– зловмисник відправляє шкідливий HTTP-запит у веб-додаток;
– створює SQL-виписку;
– задає SQL-запит в базу даних на зворотному шляху;

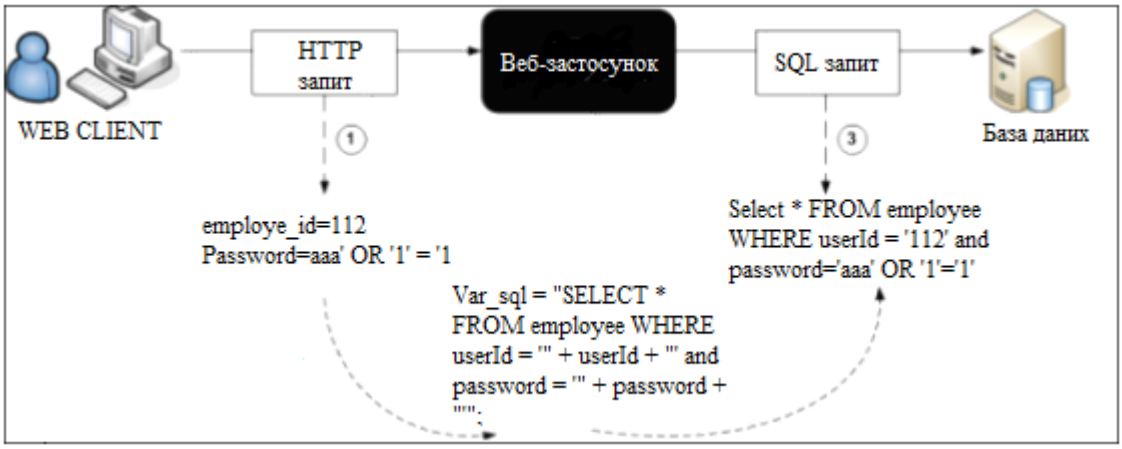


Рис. 1.1. Приклад атаки SQL-ін’єкції

Вразливі веб-додатки є основними причинами будь-якого нападу [11].

У цьому розділі будуть представлені уразливості, які природно можуть існувати в веб-додатках і можуть експлуатуватися при застосуванні атак на ін'єкцію SQL:

- недійсний вхід. Це практично найчастіша вразливість при виконанні SQLIA. У веб-додатку є деякі параметри, які використовуються у запитах SQL. Якщо їх не перевіряти, це може бути використано в атаках SQL-ін’єкцій;
- привілеї. Зазвичай в базі даних привілеї визначаються як правила для визначення того, яка суб'єкт бази даних має доступ до якого об'єкту та яку операцію асоціюють з користувачем, щоб мати можливість виконувати їх на об'єктах;
- неконтрольований змінний розмір. Якщо змінні дозволяють зберігати дані більше, ніж очікувалося, то, отже, дозволить злочинцям вводити модифіковані або фальшиві SQL-запити;
- повідомлення про помилку. Повідомлення про помилки, які генеруються заднім числом або іншими програмами на стороні сервера, можуть бути повернені на стороні клієнта та представлені в веб-браузері. Ці повідомлення не тільки корисні під час розробки для налагодження, але також підвищують ризики для програми;
- змінний орфізм. Ця змінна не повинна приймати будь-який тип даних, оскільки зловмисник може використовувати цю функцію та зберігати зловмисні дані всередині цієї змінної;
- динамічний SQL. SQL-запити, які динамічно створюються скриптами або програмами у рядку запиту. Як правило, один або більше скриптів та програм внесок і, нарешті, об'єднуючи користувацький вхід, такий як ім'я та пароль, складають пропозиції WHERE в операторі запиту;

– клієнтський контроль. Якщо верифікація вводу виконується лише на сценаріях на стороні клієнта, то функції безпеки цих сценаріїв можуть бути скасовані за допомогою кросплатформенних сценаріїв;
– процедури. Це твердження, які зберігаються в БД. Основна проблема із використанням цих процедур полягає в тому, що зловмисник може виконувати їх і пошкодити базу даних, а також операційну систему та навіть інші компоненти мережі;
– мульти вирази. Якщо база даних підтримує UNION, то атакуючий має більше шансів, тому що існує більше методів атаки для SQL-ін'єкцій;
– підзапити. Підтримка суб-вибору - слабкість для СУБД при розгляді SQL-ін'єкції.

Завдяки даним, які було досліджено у даному підрозділі можна стверджувати про необхідність вдосконалення існуючих рішень програмних засобів виявлення ін’єкцій у веб-застосунках, аналіз яких описуватимуться у п. 1.2 та вимоги визначатимуться у п. 1.3.

Таким чином постає необхідність у порівнянні програмних засобів виявлення SQL-ін’єкцій.

1.2. Аналізування програмних засобів виявлення SQL-ін’єкцій у веб-застосунках

Незважаючи на те, що розробники розгортають захисне кодування або жорсткість ОС, але їх недостатньо, щоб зупинити SQLIA для веб-додатків, тому дослідники запропонували деякі інструменти, щоб допомогти розробникам. Помітно, що існує більше підходів, які поки що не реалізовані як інструмент.

WAVES - технологія Blackbox для тестування веб-програм для уразливостей SQL-ін'єкцій. Інструмент визначає всі точки веб-додатків, які можуть бути використані для введення SQLIA [3].

JDBC-Checker [4, 5] не розроблявся з метою виявлення та запобігання загальних SQLIA, але може бути використаний для запобігання атак, які використовують невідповідність типів у динамічно створеному ланцюжку запитів.

У запитах SQL Guard та SQL Check перевіряються під час виконання, виходячи з моделі, яка виражається як граматика, яка приймає тільки легальні запити. SQL Guard розглядає структуру запиту до і після додавання введення користувача на основі моделі.

AMNESIA поєднує в собі статичний аналіз та моніторинг виконання. У статичній фазі він створює моделі різних типів запитів, які програма може створювати на законних підставах у кожній точці доступу до бази даних. Запити перехоплюються, перш ніж вони надсилаються в базу даних та перевіряються на статично побудовані моделі в динамічній фазі.

WebSSARI використовує статичний аналіз, щоб перевірити потоки завад від попередніх умов для чутливих функцій. Він працює на основі дезинфікованого входу, який пройшов через визначений набір фільтрів. Обмеження підходу є адекватними передумовами для чутливої функції не можна точно виразити, тому деякі фільтри можуть бути опущені.

SecuriFly - це ще один інструмент, який був реалізований для java. Незважаючи на інший інструмент, переслідує рядок замість символу для недоступної інформації. SecurityFly намагається дезінфікувати рядки запитів, які були згенеровані за допомогою введеного вкладу, але, на жаль, ін'єкція в числових полях не може зупинитися за допомогою цього підходу. Труднощі виявлення всіх джерел вводу користувача є основним обмеженням цього підходу.

Позитивне затухання не тільки фокусується на позитивному заглушенні, а не на негативному заглушенні, але також автоматично і потребує розробника втручання. Крім того, цей підхід вигідний від оцінки

синтаксису, який дає розробникам механізм регулювання використання рядкових даних, заснованого не тільки на його джерелі, але також на його синтаксичній ролі в рядку запиту.

IDS використовує систему виявлення вторгнень (IDS) для виявлення SQLIA, засновану на техніці навчання комп'ютера. Технологія створює моделі типових запитів, а потім під час виконання, запити, які не відповідають моделі, будуть ідентифіковані як атаки. Цей інструмент успішно виявляє атаки, але це серйозно залежить від тренувань. Іншими словами, буде сформовано багато помилкових та ложних негативів.

Іншим підходом у цій категорії є SQL-IDS, який зосереджується на написанні специфікацій для веб-додатки, які описують передбачувану структуру SQL-запитів, які виробляються додатком, а також автоматичного контролю за виконанням цих SQL-запитів щодо порушень цих специфікацій.

SQLPrevent складається з перехоплювача запиту HTTP. Початковий потік даних змінюється, коли SQLPrevent розгортається на веб-сервері. HTTP-запити зберігаються в поточному поточному локальному сховищі.

У табл. 1.1 узагальнюють результати цього порівняння. Символ "+" використовується для інструмента, який може успішно припинити всі атаки цього типу. Символ "-" використовується для інструмента, який не здатний зупинити атаки цього типу. Символ "*" означає інструмент, який тип атаки лише частково обумовлений природними обмеженнями основного підходу.

Як показано в табл. 1.1, збережена процедура є критичною атакою, яка деяким інструментам важко її зупинити. Він складається з запитів, які можуть виконуватись у базі даних. Однак більшість інструментів враховують лише ті запити, які генеруються всередині програми. Отже, цей тип атаки створює серйозні проблеми для деяких інструментів.

Таблиця 1.1

Порівняння інструментів щодо типів атак

	SQL IDS	Swaddler	ID S	CANDID	AMNE SIA	SQL Check	SQL Query	Web SSARI	SecuriFly	WAVES	Possitive taining	SQL Prvent
Тавтологія	+	*	*	*	+	+	+	+	*	*	+	+
Некоректність	+	*	*	*	+	+	+	+	*	*	+	+
Комбінації	+	*	*	*	+	+	+	+	*	*	+	+
Union	+	*	*	*	+	+	+	+	*	*	+	+
Процедури	+	*	*	*	-	-	-	+	*	*	+	+
Вивід	+	*	*	*	+	+	+	+	*	*	+	+
Кодування	+	*	*	*	+	+	+	+	*	*	+	+

1.3.Визначення вимог до програмного застосунку виявлення SQL-ін’єкцій у веб-застосунках

На рис. 1.2 зображена графічна схема виявлення SQL-ін’єкцій у веб-застосунках.

Також можна побачити програмне забезпечення, який буде реалізований у рамках магістерської дисертації, що виділеий червоною лінією, який містить [7]:

- модуль збору та анілізу подій;
- модуль відображення даних про вторгнення;
- модуль для зберігання даних у базу даних.

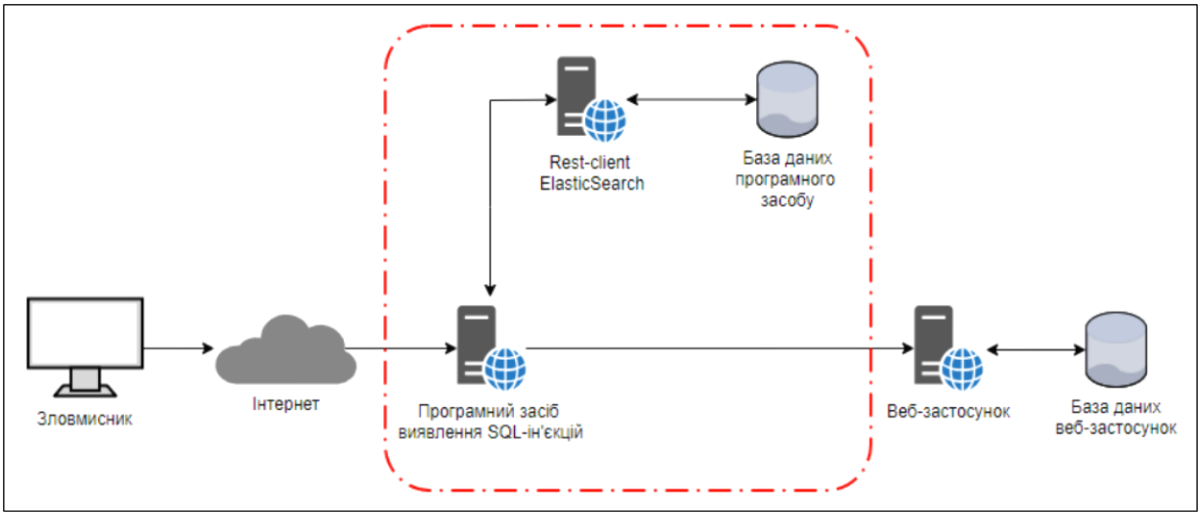


Рис.1.2. Графічна схема програмного забезпечення для виявлення SQL-ін’єкцій у веб-застосунках

Основні функціональні вимоги, що ставляться до серверної та клієнтської частин системи, наведено в табл. 1.1 та 1.2.

Для визначення коректності вимог в цілому для системи необхідно встановити чи не суперечать вони одна одній та чи не перекриваються між собою. Матриці залежності вимог до програмного забезпечення для виявлення SQL-ін’єкції у веб-застосунку наведені в табл. 1.4 та 1.5.

Таблиця 1.2

Функціональні вимоги серверної частини, що ставляться до системи

Кодовий номер	Вимога
В1	Можливість виявляти SQL-ін’єкції у веб-застосунку
В2	Запис інформації про SQL-ін’єкцію до лог-файлу
В3	Повідомлення про SQL-ін’єкцію адміністратору веб-застосунку
В4	Запис інформації про SQL-ін’єкцію до бази даних
В5	Безперервна робота програмного засобу

Продовження табл. 1.2

В6	Можливість пошуку SQL-ін’єкцій

Таблиця 1.3

Функціональні вимоги клієнтської частини, що ставляться до системи

Кодовий номер	Вимога
В7	Можливість перегляду стану системи
В8	Можливість перегляду налаштувань системи
В9	Можливість вивантаження таблиці стану системи у Excel
В10	Можливість авторизації у системі
В11	Можливість змінювати налаштування програмного забезпечення виявлення SQL-ін’єкції у веб-застосунку

Таблиця 1.4

Матриця залежності функціональних вимог до серверної частини програмного забезпечення

	В1	В2	В3	В4	В5	В6
В1	X	X	X	X	X	X
В2		X	X	X	X	X
В3			X	X	X	X
В4				X	X	X
В5					X	X
В6						X

Таблиця 1.5

Матриця залежності функціональних вимог до клієнтської частини програмного забезпечення

	В7	В8	В9	В10	В11
В7	X	X	X	X	X
В8		X	X	X	X
В9			X	X	X
В10				X	X
В11					X

Як видно з матриць залежності, які наведені у таблицях 1.4 та 1.5 вимог жодних протиріч або накладань вимог не виявлено.

1.4.Висновки

У цьому розділі представилено різні типи SQLIA. Досліджено програмні засоби виявлення SQL-ін'єкцій у веб-застосунках. Після цього проведено порівняння цих програмних засобів з точки зору їх здатності виявляти SQLIA. Крім того, поточні інструменти порівнювалися на основі вимог до розгортання (модифікації вихідного коду, додаткової інфраструктури та автоматизації виявлення або запобігання) та загальних параметрів оцінки (ефективність, ефективність, стабільність, гнучкість та продуктивність).

Основні результати розділу опубліковано в роботі [10].

2. КОНЦЕПТУАЛЬНА МОДЕЛЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЯВЛЕННЯ SQL-ІН'ЄКЦІЙ У ВЕБ-ЗАСТОСУНКАХ

2.1.Формалізування процесу виявлення SQL-ін'єкцій у веб-застосунках

Для спрощення розуміння предметної області та завдань, які ставляться перед розроблюваним програмним засобом, необхідно змоделювати його з метою його формалізації та опису. Для цього рекомендується використати методику функціонального моделювання та графічного опису процесів IDEF0 [12].

IDEF0 – методологія функціонального моделювання і графічного описання процесів, призначена для формалізації і опису процесів та систем. Особливістю IDEF0 є її акцент на ієрархічне представлення об'єктів, що значно полегшує розуміння предметної області [13].

Діяльність програмного засобу представлено за допомогою діаграми А-0 в нотації IDEF0 та декомпозиції її на окремі функції за допомогою діаграм А0 та А1 в нотації IDEF0. Дані діаграми наведено на рис. 2.1. та 2.2 [10].

Діаграма А0 демонструє найбільш абстрактний опис програмного засобу, на якому вказаний сам суб'єкт моделювання, цілі, які необхідно досягнути, вхідні дані, вихідні дані та механізми – ресурси, що виконують роботу. Діаграма А0 є декомпозицією контекстної діаграми, вона виділяє основні функції системи, вказує потоки даних між ними, вхідні та вихідні дані для різних функцій, які даними функціями можуть керувати. Діаграма А1 є декомпозицією блоку А1 з діаграми А0, показує дії необхідні для виконання процесу А1 – «Виявлення SQL-ін'єкцій у веб-застосунках». На основі найнижчого рівня декомпозиції можливо зробити висновок для формування необхідної архітектури програмного засобу та його конструювання [10].

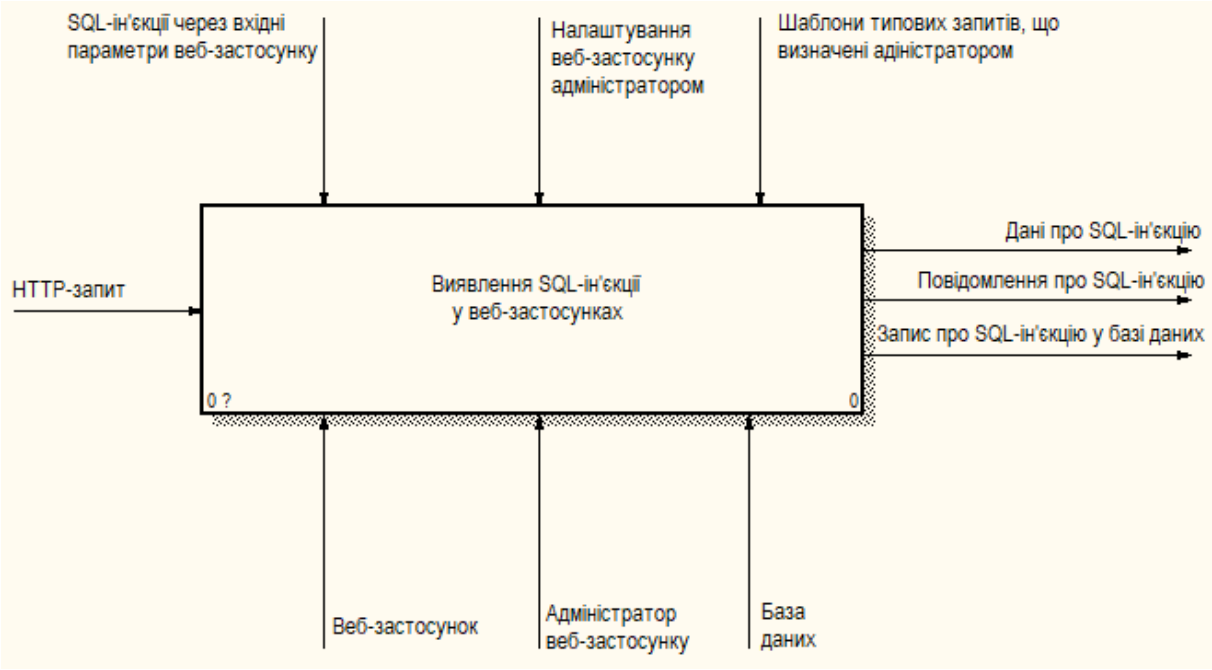


Рис. 2.1. Функціональна модель програмного засобу

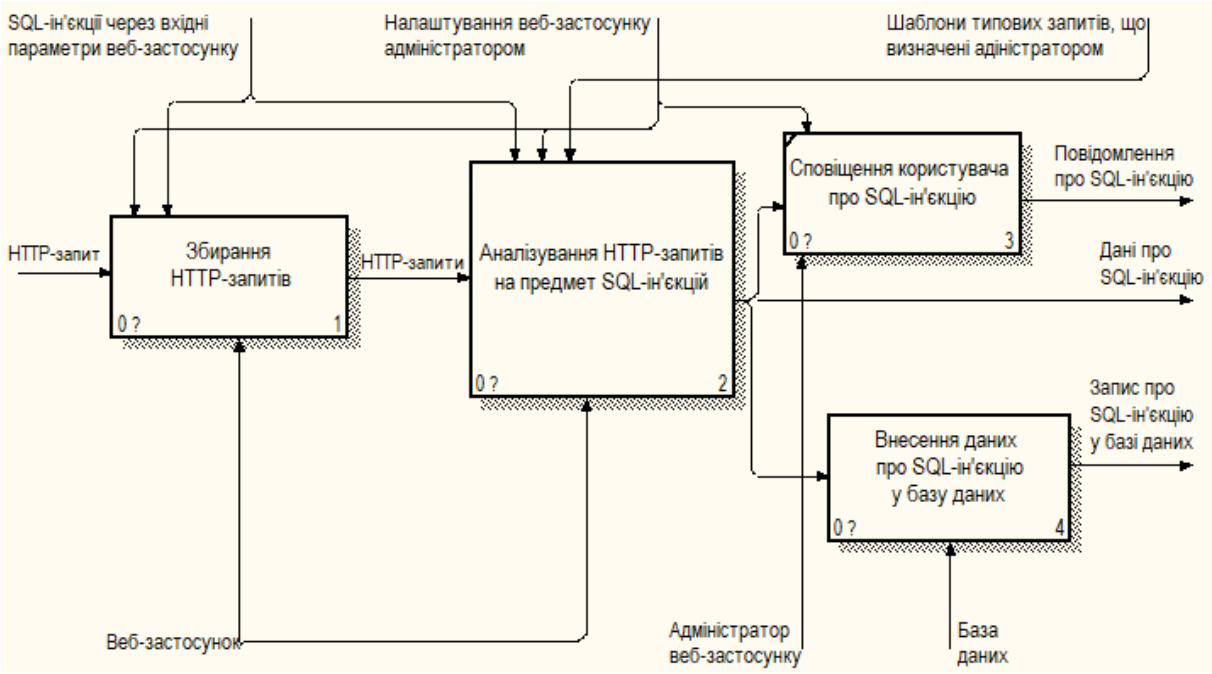


Рис. 2.2. Декомпозиція функціональної моделі програмного засобу

Згідно зі структурою програмного засобу виявлення SQL-ін’єкцій, яка була досліджена у підрозділі 1.1 першого розділу, на рис. 2.2 можна побачити головні процеси, які виконуються при роботі програмного засобу [10]:

– збирання HTTP-зіпитів – на цьому процесі виконується збір

первинної інформації про запити, які надсилаються до веб-застосункку за допомогою підсистеми збору;

– аналізування HTTP-запитів на предмет SQL-ін’єкцій – на цьому процесі здійснюється пошук та аналіз HTTP-запитів, які не відповідають шаблонам типових запитів, що визначені адміністратором веб-застосунку, за допомогою підсистеми аналізу;
– сповіщення користувача про SQL-ін’єкцію – процес, на якому здійснюється відправка повідомлення про SQL-ін’єкцію на пошту користувача за допомогою підсистеми представлення даних, що дозволяє адміністраторам програмного засобу слідкувати за станом веб-застосунку;
– внесення даних про SQL-ін’єкцію у базу даних – процес, на якому відбувається записи та збереження інформації про SQL-ін’єкцію у базу даних та лог-файли веб-застосунку, що захищається.

Вхідним даними є «HTTP-запити» – це сукупність запитів, які надсилалися у веб-застосунок. Також на схемі структури бізнес-процесів можна побачити механізми роботи програмного засобу виявлення SQL-ін’єкцій:

– програмне забезпечення виявлення SQL-ін’єкцій, який займається аналізом використання ввірених їм мережевих ресурсів і, у випадку виявлення яких-небудь підозрілих або просто нетипових подій, здатний робити деякі самостійні дії по виявленню, ідентифікації та усунення їх причини;
– адміністратор веб-застосунку – механізм, який відповідає за контроль роботи програмного забезпечення, та її взаємодії з системою;
– база даних – механізм, який відповідає за зберігання та отримання даних;

Вихідними даними є так звана «перетворена інформація»:

– дані про SQL-ін’єкцію – форматовані дані, які були отримані за допомогою підсистеми аналізу;
– повідомлення про SQL-ін’єкцію – сформоване текстове повідомлення, яке було надіслано підсистемою сповіщення про SQL-ін’єкцію з вже готових «даних про SQL-ін’єкцію», які були отримані із підсистеми аналізу;
– запис про SQL-ін’єкцію у базі даних – запис, які створений підсистемою збереження інформації у базі даних.

Умови та обмеження програмного засобу також можна побачити на рис. 2.1 та 2.2 – це внутрішні правила програмного засобу виявлення SQL-ін’єкцій у веб-застосунках.

Для більш детального опису основного процесу виявлення SQL-ін’єкцій була побудована діаграма у нотації IDEF зображна на рис. 2.3, що є декомпозицією блоку з діаграми A0, показує дії необхідні для виконання процесу A1 – «Аналізування HTTP-запитів на предмет SQL-ін’єкцій».

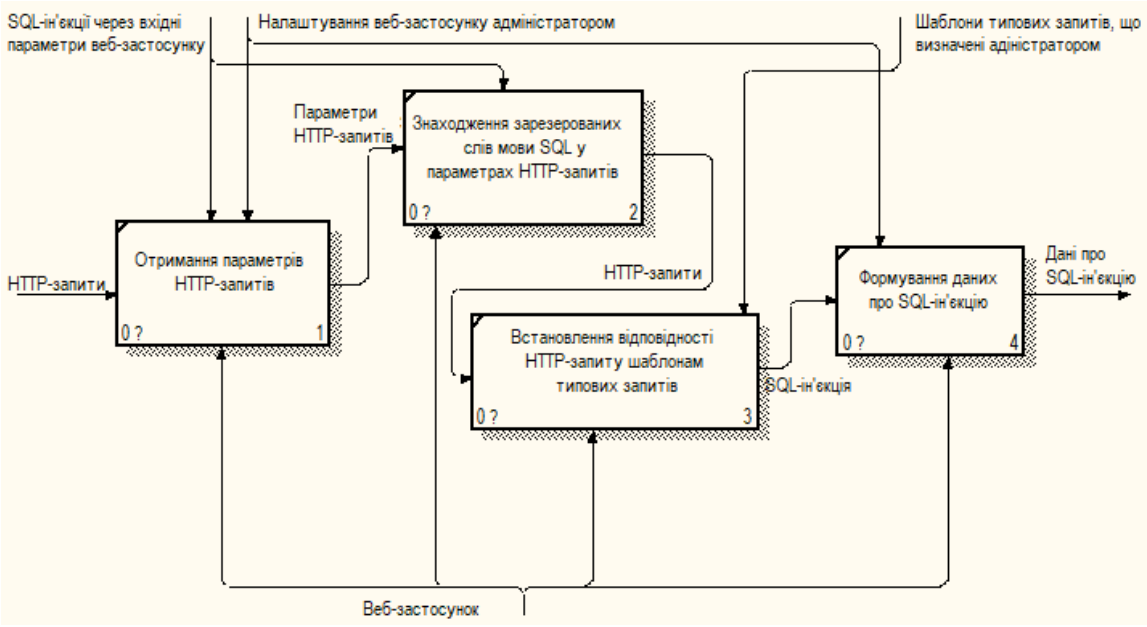


Рис. 2.3. Декомпозиція функції «Аналізування HTTP-запитів на предмет SQL-ін’єкцій»

Даної моделі достатньо для розуміння процесів, що лежать в основі роботи програмного засобу для виявлення SQL-ін’єкцій.

Вхідним даними є «HTTP-запити». Також на схемі структури бізнес-процесів можна побачити механізми роботи програмного засобу виявлення SQL-ін’єкцій:

– програмне забезпечення виявлення SQL-ін’єкцій, який займається аналізом використання ввірених їм мережевих ресурсів і, у випадку виявлення яких-небудь підозрілих або просто нетипових подій, здатний робити деякі самостійні дії по виявленню, ідентифікації та усунення їх причини.

Вихідними даними є так звана «перетворена інформація»:

– дані про SQL-ін’єкцію – форматовані дані, які були отримані за допомогою підсистеми аналізу.

Також на рис. 2.3 можна побачити головні процеси, які виконуються при роботі програмного засобу:

– отримання параметрів HTTP-зіпитів – на цьому процесі виконується збирання вхідних параметрів HTTP-запитів, які надсилаються до веб-застосункку за допомогою підсистеми збору;
– знаходження зарезервованих слів мови SQL у параметрах HTTP-запитів – на цьому процесі здійснюється знаходження зарезервованих слів у параметрів HTTP-запитів, та якщо такі слова були знайдені, то данний запит передається на наступний процес та перевіряється відповідність запиту шаблонам типових запитів, що визначені адміністратором веб-застосунку;
– встановлення відповідності HTTP-запиту шаблонам типових запітв – процес, на якому здійснюється перевіряння HTTP-запиту на відповідність шаблонам типових запитів визначених адміністратором веб-застосунку, та якщо запит не буде

відповідати шаблонам, то такий запит буде вважатися SQL-ін’єкцією;

– Формування даних про SQL-ін'єкцію – процес, на якому здійснюється формування інформації про SQL-ін’єкцію, для передачі цих даних до процесу сповіщення користувача про ін’єкцію та збереження до бази даних.

2.2.Визначення зв’язків між етапами виявлення SQL-ін’єкцій у веб-застосунках

Для визначення зв’язків між етапами роботи програмного засобу було використано методику функціонального моделювання та графічного опису процесів IDEF3, який представляє механізм документування та збору інформації про процес виявлення SQL-ін’єкцій [14].

IDEF3 є стандартом документування технологічних процесів, що відбуваються у системі, і надає інструментарій для наочного дослідження і моделювання їх сценаріїв. Сценарієм ми називаємо опис послідовності змін властивостей об'єкта, в рамках даного процесу. Виконання кожного сценарію супроводжується відповідним документообігом, який складається з двох основних потоків: документів, що визначають структуру і послідовність процесу (технологічних вказівок, описів стандартів і т.д.), і документів, що відображають хід його виконання (результатів тестів і експертиз, і т.д.). Для ефективного управління будь-яким процесом, необхідно мати детальне уявлення про його сценарії і структурі супутнього документообігу. Засоби документування та моделювання IDEF3 дозволяють виконувати наступні завданнях [15]:

– документувати наявні дані про технології процесу;
– визначати і аналізувати точки впливу потоків супутнього документообігу на сценарій технологічних процесів;
– визначати ситуації, в яких потрібно прийняття рішення, що впливає на життєвий цикл процесу;

– сприяти прийняттю оптимальних рішень при реорганізації технологічних процесів.

Система описується як упорядкована послідовність подій з одночасним описом об'єктів, що мають відношення до процесу.

На рис. 2.4 можна побачити декомпозицію контекстної діаграми (рис. 2.1), на якій зображені зв’язки між процесами «Виявлення SQL-ін'єкції у веб-застосунках» [10].

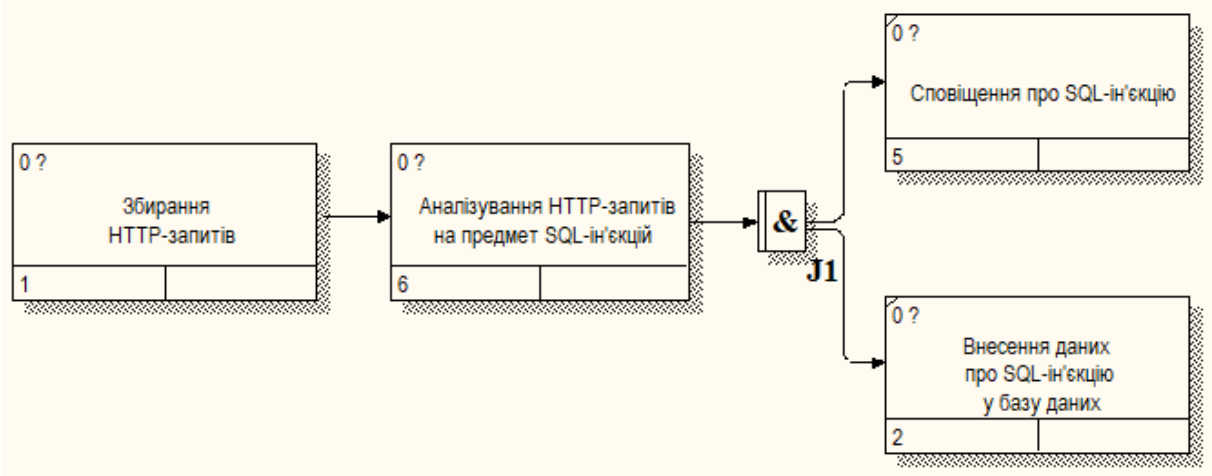


Рис. 2.4. Декомпозиція контекстної діаграми в нотації IDEF3

На рис. 2.5 та 2.6 зображені зв’язки між етапами роботами процесів «Виявлення SQL-ін'єкції у веб-застосунках» [10]:

– «Збирання HTTP-запитів»;
– «Аналізування HTTP-запитів на предмет SQL-ін’єкцій»;
– «Сповіщення про SQL-ін’єкцію»;
– «Внесення даних про SQL-ін’єкцію у базу даних».

Так, як основні дії з даними, а саме збір та обробка інформації виконується у блоках «Збирання HTTP-запитів» та «Аналізування HTTP-запитів на предмет SQL-ін’єкцій», тому розглянемо декомпозиції саме цих робіт у нотації IDEF3.

Процес «Збирання HTTP-запитів», що зображений на рис. 2.4, акумулює дані про роботу веб-застосунку. На цьому етапі роботи

програмного засобу виявлення SQL-ін’єкцій підсистема збору фільтрує мережеві потоки для того, щоб допускати, відмовляти, шифрувати, пропускати через проксі весь комп'ютерний трафік згідно з набором правил та інших критеріїв. Далі, якщо мережевий екран знаходить HTTP-запити, що не відповідають вимогам системи, користувацьким налаштуванням та правилам безпеки, ця подія записується у системний лог-файл, та цикл збору інформації починається спочатку.

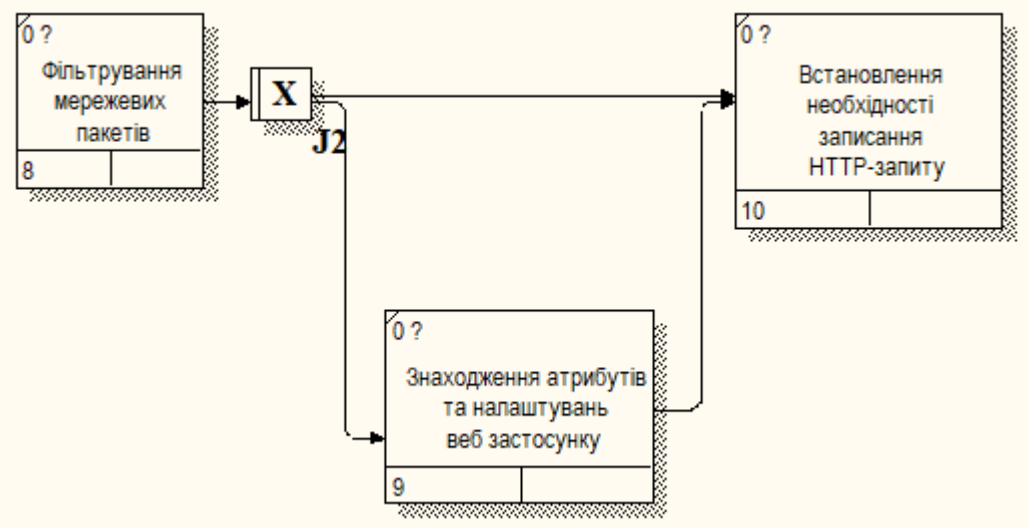


Рис. 2.5. Схема структури з’язків процесу «Збирання HTTP-запитів»

Процес «Аналізування HTTP-запитів на предмет SQL-ін’єкцій», зображений на рис. 2.6, виконує пошук SQL-ін’єкцій відповідного типу. Вхідними даними для аналізатора являється інформація із підсистеми збору інформації. Відповідно до користувацьких правил та налаштувань захисту системи аналізатор формує звіт про вторгнення. Якщо адміністратором не було налаштовано правила аналізу подій, то програмний застосунок формує звіт по усім HTTP-запитам, які відносяться до даної атаки.

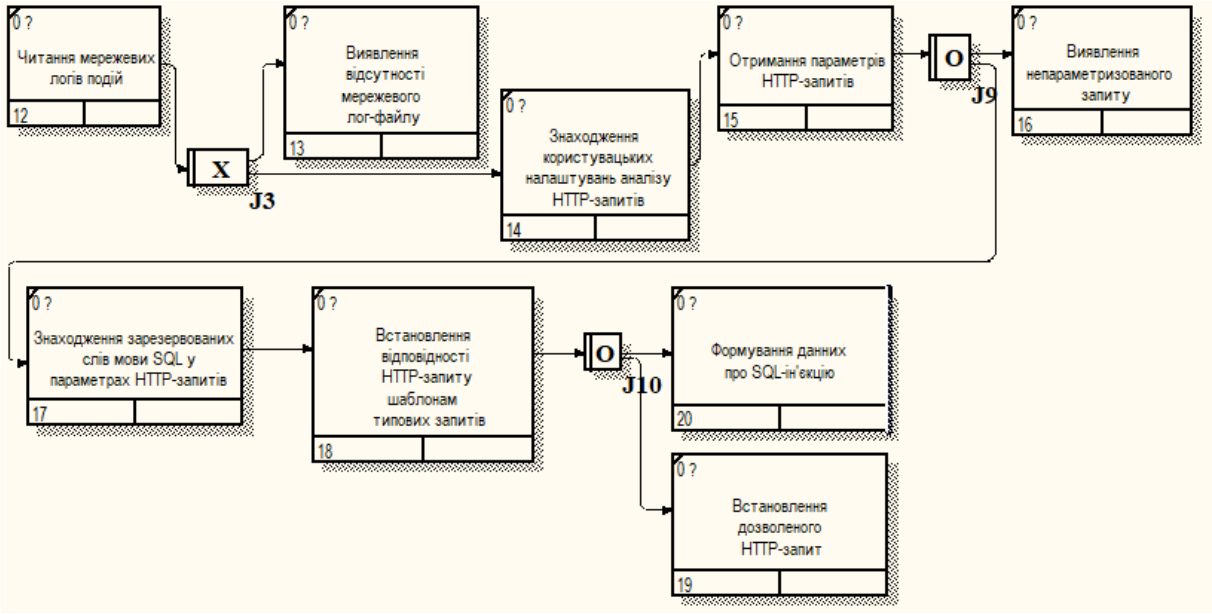


Рис. 2.6. Декомпозиція роботи «Аналізування HTTP-запитів на предмет SQL-ін’єкцій»

Після сформованого звіту даних про SQL-ін’єкцій у процесі «Аналізування HTTP-запитів на предмет SQL-ін’єкцій», інформація передається до процесів «Сповіщення про SQL-ін'єкцію» та «Внесення даних про SQL-ін'єкцію у базу даних», що є підсистемами відображення та збереження інформації відповідно.

2.3.Визначення потоків даних між етапами виявлення SQL-ін’єкцій у веб-застосунках

Моделювання потоків даних – один з основних інструментів структурного аналізу і проектування інформаційних систем.

Для визначення потоків даних програмного засобу виявлення SQL-ін’єкцій було використано методику графічного структурного аналізу – DFD (див. рис. 2.7), що описує зовнішні по відношенню до системи джерела і адресати даних, логічні функції, потоки даних і сховища даних, до яких здійснюється доступ [16].

Інформаційна система приймає ззовні потоки даних. Для позначення елементів середовища функціонування системи використовується поняття

зовнішньої сутності. Усередині системи існують процеси перетворення інформації, які породжують нові потоки даних. Потоки даних можуть надходити на вхід до інших процесів, поміщатися (і вилучатись) в накопичувачі даних, передаватися до зовнішніх сутностей.

Модель DFD, як і більшість інших структурних моделей – ієрархічна модель. Кожен процес може бути підданий декомпозиції, тобто розбиття на структурні складові, відносини між якими в тій же нотації можуть бути показані на окремій діаграмі. Коли досягнута необхідна глибина декомпозиції – процес нижнього рівня супроводжується так званою міні-специфікацією (текстовим описом) [17].

Крім того, нотація DFD підтримує поняття підсистеми структурного компонента розроблюваного засобу.

Нотація DFD – зручний засіб для формування контекстної діаграми, тобто діаграми, яка б показала, що розробляється в комунікації із зовнішнім середовищем. Це – діаграма верхнього рівня в ієрархії діаграм DFD. Її призначення – обмежити рамки системи, визначити, де закінчується розробляється система і починається середовище.

На рис. 2.7 зображена декомпозиція контекстної діаграми, що зображена на рис. 2.1. Центральним процесом є «Виявлення SQL-ін'єкції у веб-застосунках». На його вхід поступає HTTP-запит. Виходом з цього процесу будуть записи у лог-файлі, записи у базі даних, повідомлення відправлене на пошту та дані про SQL-ін’єкцію. Сховищами даних виступають [10]:

– лог-файл;
– база даних;
– список HTTP-запитів.

Також на діаграмі зображено дочірні процеси, які відображають потоки даних між ними, а саме:

– «Збирання HTTP-запитів»;
– «Аналізування HTTP-запитів на предмет SQL-ін’єкцій»;

– «Сповіщення про SQL-ін’єкцію»;
– «Внесення даних про SQL-ін’єкцію у базу даних».

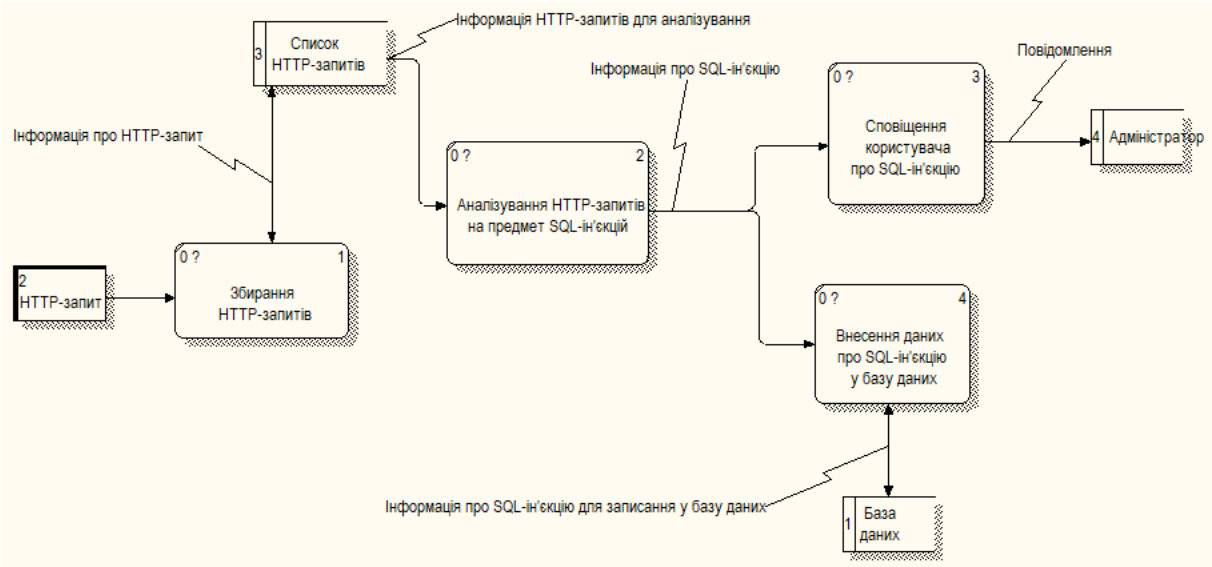


Рис. 2.7. Декомпозиція контекстної діаграми в нотації DFD

На рис. 2.8 зображена схема структури потоків даних програмного засобу виявлення SQL-ін’єкцій у веб-застосунках. На схемі можемо бачити, що потоком даних є «дані про вторгнення».

Також на схемі зображено дочірні процеси [10]:

– «Отримання параметрів HTTP-запитів»;
– «Знаходження зарезервованих слів мови SQL у параметрах HTTP-запитів»;
– «Встановлення відповідності HTTP-запиту шаблонам типових запитів»;
– «Формування даних про SQL-ін’єкцію».

Дані процеси використовують інформацію про вторгнення, що були сформовані в підсистемі аналізу, та записуються у сховища даних програмного засобу виявлення SQL-ін’єкцій [10]:

– список параметрів HTTP-запитів;
– довідник зарезервованих слів мови SQL;
– шаблони типових запитів.

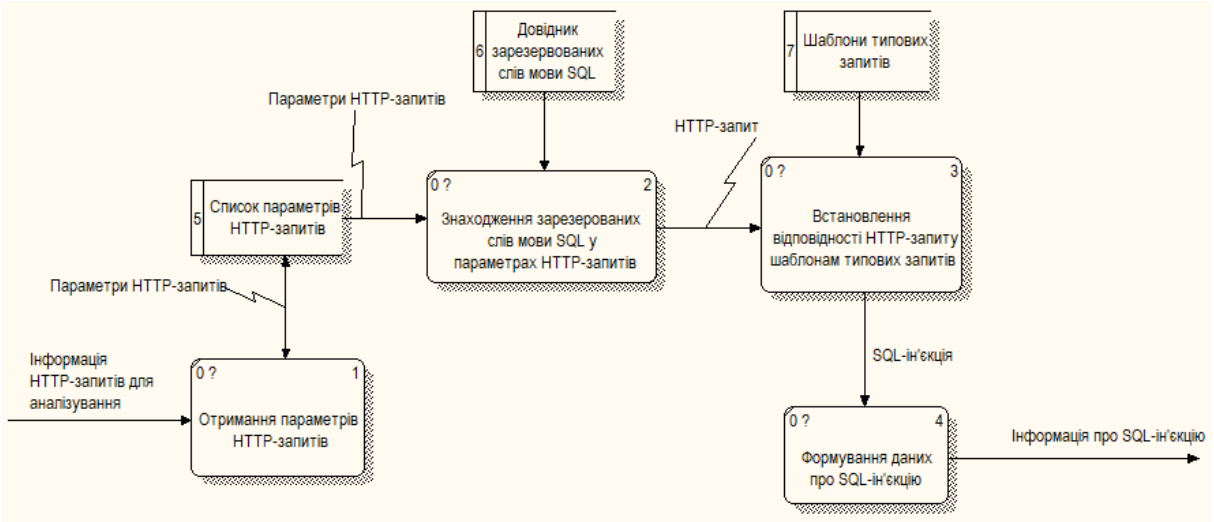


Рис. 2.8. Схема структури потоків даних

І як результат, після виконання цих процесів дані зберігаються у лог-файлі та базі даних.

2.4. Висновки

Для спрощення розуміння предметної області та завдань, які ставляться перед розроблюваним програмним забезпеченням, на функцільному рівні концептуальної моделі, виконано його моделювання у графічній нотації IDEF0 і, як наслідок,формалізовано програмне забезпечення.

На процесному рівні концептуальної моделі для визначення зв’язків між етапами роботи програмного забезпечення виконане моделювання у графічній нотації IDEF3. Це підхід дозволив задокументувати та зібрати інформацію про процес виявлення SQL-ін’єкцій, визначити черговість його етапів.

Для визначення потоків даних програмного забезпечення виявлення SQL-ін’єкцій на рівні потоків даних концептуальної моделі виконано моделювання програмного забезпечення у графічній нотації DFD, що описує зовнішні стосовно програмного забезпечення джерела і адресати даних, логічні функції, потоки даних і сховища даних, до яких здійснюється доступ.

Таким чином, концептуальне моделювання дозволило формалізувати роботу програмного засобу набором функцій. Тоді як завдяки побудові процесної моделі та моделі потоків даних описано взаємозв’язки між етапами оповіщення про SQL-ін’єкцію та обмін даних між ними. Зокрема, це дозволить побудувати об’єктно-орієнтовану модель програмного засобу виявлення SQL-ін’єкцій у веб-застосунках.

Основні результати розділу опубліковано в роботі [10].

3. ОБ'ЄКТНО-ОРІЄНТОВАНА МОДЕЛЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЯВЛЕННЯ SQL-ІН'ЄКЦІЙ У ВЕБ-ЗАСТОСУНКАХ

3.1. Визначення варіантів використання програмного забезпечення

Для проектування варіантів використання необхідно визначити дійових осіб – акторів. Оскільки програмне забезпечення виявлення вторгнень ставить на меті дати можливість користувачеві відслідковувати та виявляти SQL-ін'єкції у веб-застосунку, єдиним учасником системи є адміністратор (див. рис. 3.1) [10, 18].

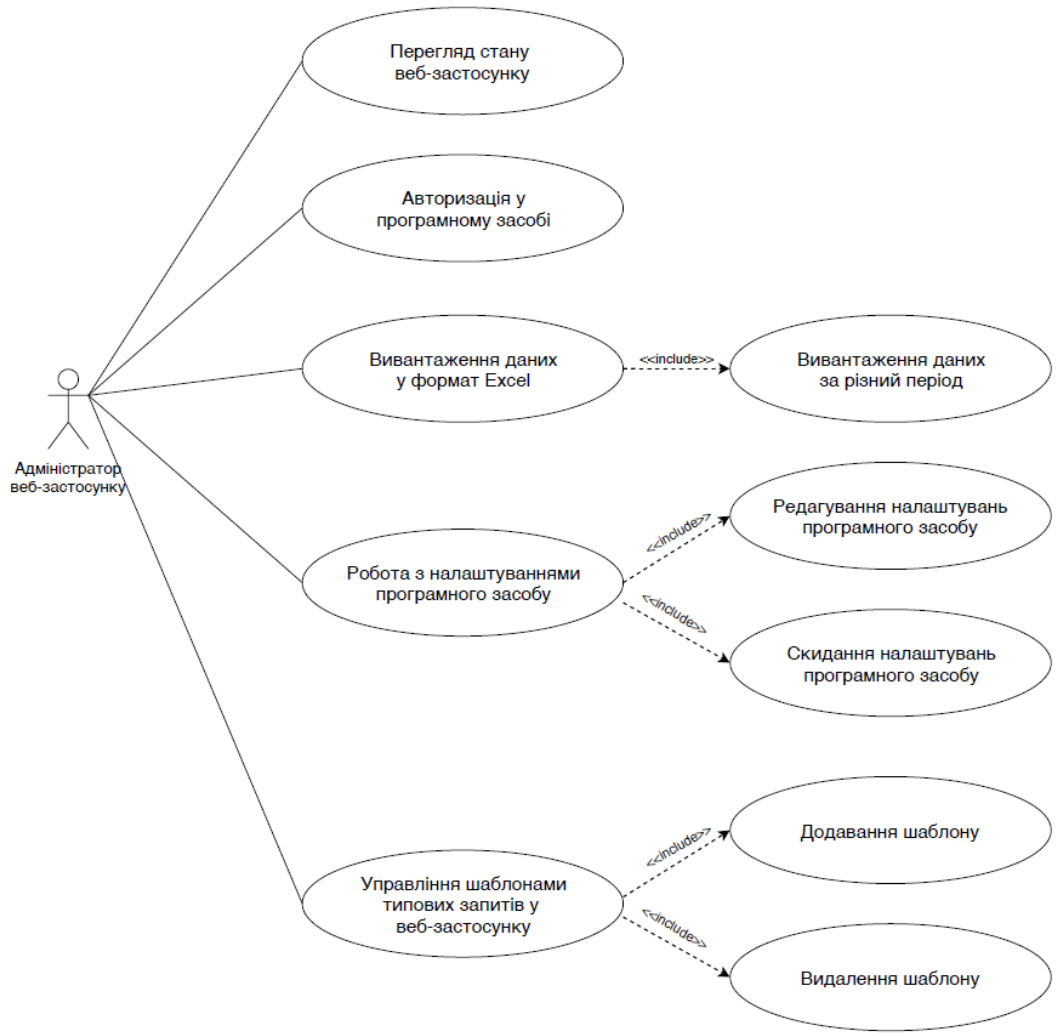


Рис. 3.1. Структурна схема варіантів використання програмного забезпечення для виявлення SQL-ін'єкцій

Детальний опис кожного з варіантів використання наведено в табл. 3.1 – 3.8.

Таблиця 3.1

Варіант використання «Авторизація на сторінці перегляду стану веб-застосунку»

Найменування	Авторизація на сторінці перегляду стану веб-застосунку
Первинний актор	Адміністратор
Інші актори	Немає
Опис	Можливість авторизуватись на сторінці перегляду стану системи
Попередні умови	Відкрита сторінка перегляду стану веб-застосунку
Вихідні умови	Відкрита сторінка перегляду стану веб-застосунку

Таблиця 3.2

Варіант використання «Створення налаштувань програмного засобу виявлення SQL-ін’єкцій»

Найменування	Створення налаштувань програмного засобу виявлення SQL-ін’єкцій
Первинний актор	Адміністратор
Інші актори	Немає
Опис	Можливість налаштувати програмне забезпечення виявлення SQL-ін’єкцій
Попередні умови	Відкрита сторінка налаштувань програмного засобу виявлення SQL-ін’єкцій та адміністратор авторизований

Продовження табл.3.2

Вихідні умови	Прийняті налаштування програмного засобу виявлення SQL-ін’єкцій

Таблиця 3.3

Варіант використання «Вивантаження даних про SQL-ін’єкції у форматі Excel»

Найменування	Вивантаження даних про SQL-ін’єкції у форматі Excel
Первинний актор	Адміністратор
Інші актори	Немає
Опис	Можливість вивантаження даних про виявлені SQL-ін’єкції у веб-застосунку форматі Excel
Попередні умови	Відкрита сторінка перегляду виявлених SQL-ін’єкцій
Вихідні умови	Сформований файл у форматі Excel

Таблиця 3.4

Варіант використання «Видалити користувацькі налаштування програмного засобу виявлення SQL-ін’єкцій»

Найменування	Видалити користувацькі налаштування програмного засобу виявлення SQL-ін’єкці
Первинний актор	Адміністратор
Інші актори	Немає
Опис	Можливість видалити налаштування програмного засобу

Продовження табл. 3.4

Попередні умови	Відкрита сторінка налаштувань прогрмного засобу виявлення SQL-ін’єкцій та адміністратор авторизований
Вихідні умови	Видалено користувацькі налаштування системи виявлення вторгнень

Таблиця 3.5

Варіант використання «Редагувати користувацькі налаштування системи виявлення вторгнень»

Найменування	Редагувати користувацькі налаштування системи
Первинний актор	Користувач
Інші актори	Немає
Опис	Можливість редагувати налаштування системи
Попередні умови	Відкрита сторінка налаштування системи
Вихідні умови	Відредаговані налаштування системи виявлення вторгнень

Таблиця 3.6

Варіант використання «Перегляд стану веб-застосунку»

Найменування	Перегляд стану веб-застосунку
Первинний актор	Адміністратор
Інші актори	Немає
Опис	Можливість перегляду стану веб-застосунку
Попередні умови	Відкрита сторінка перегляду стану веб-застосунку та адміністратор авторизований

Продовження табл. 3.6

Вихідні умови	Перегляд стану веб-застосунку

Таблиця 3.7

Варіант використання «Отримати повідомлення на пошту адміністратора про SQL-ін’єкцію»

Найменування	Отримати повідомлення на пошту адміністратора про SQL-ін’єкцію
Первинний актор	Адміністратор
Інші актори	Немає
Опис	Можливість отримати повідомлення на пошту про SQL-ін’єкцію
Попередні умови	Запущений програмне забезпечення виявлення SQL-ін’єкцій
Вихідні умови	Відредаговані налаштування програмного засобу виявлення SQL-ін’єкцій

Таблиця 3.8

Варіант використання «Управління шаблонами типових запитів у веб-застосунку»

Найменування	Управління шаблонами типових запитів у веб-застосунку
Первинний актор	Адміністратор
Інші актори	Немає
Опис	Можливість створення та видалення шаблонів типових запитів у веб-застосунку
Попередні умови	Відкрита сторінка для роботи з шаблонами типових запитів у веб-застосунку

Продовження табл.3.8

Вихідні умови	Створений або видалений шаблон типового запиту у веб-застосунку

3.2. Визначення логічної структури програмного забезпечення

Логічна структура містить набір функціонально-логічних модулів, що включають процедури і об'єкти, що представляють собою стандартні прототипи додатків баз даних: форми, вікна для перегляду таблиць бази даних, звіти запити і т.д. і оригінальні програмні одиниці, що реалізують деяку автоматизуються функцій або завдання досліджуваної предметної області [19].

На рис. 3.2 – 3.3 зображено структурну схему класів програмного засобу, яка представлена за допомогою відповідної UML-діаграми [10, 20].

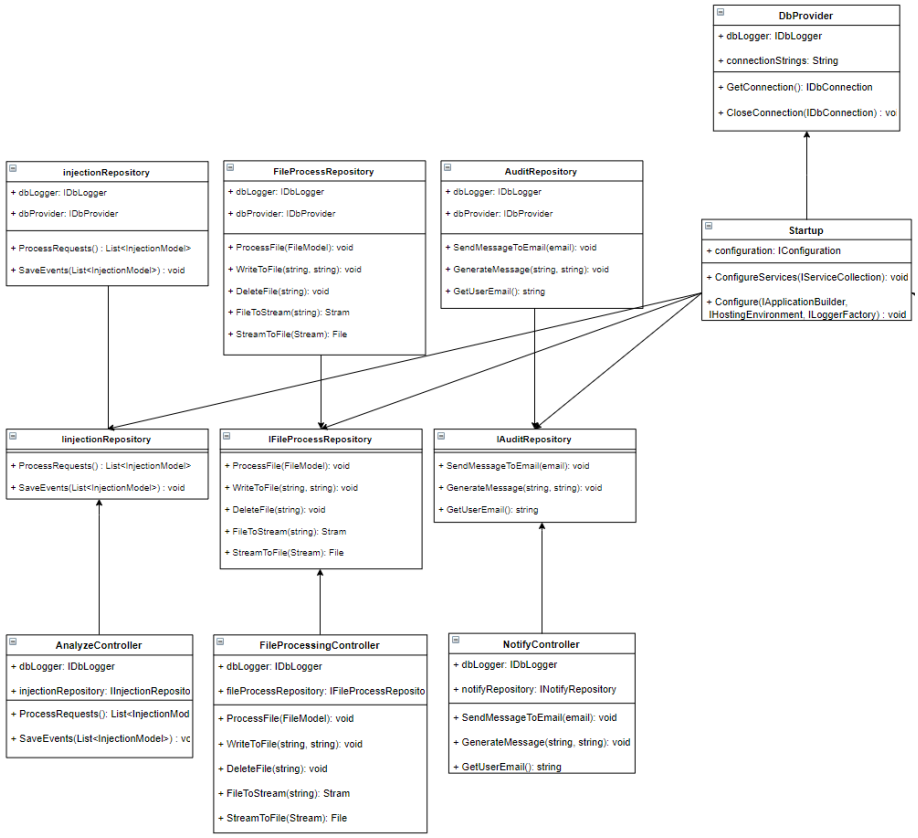


Рис. 3.2 Структурна схема класів програмного засобу

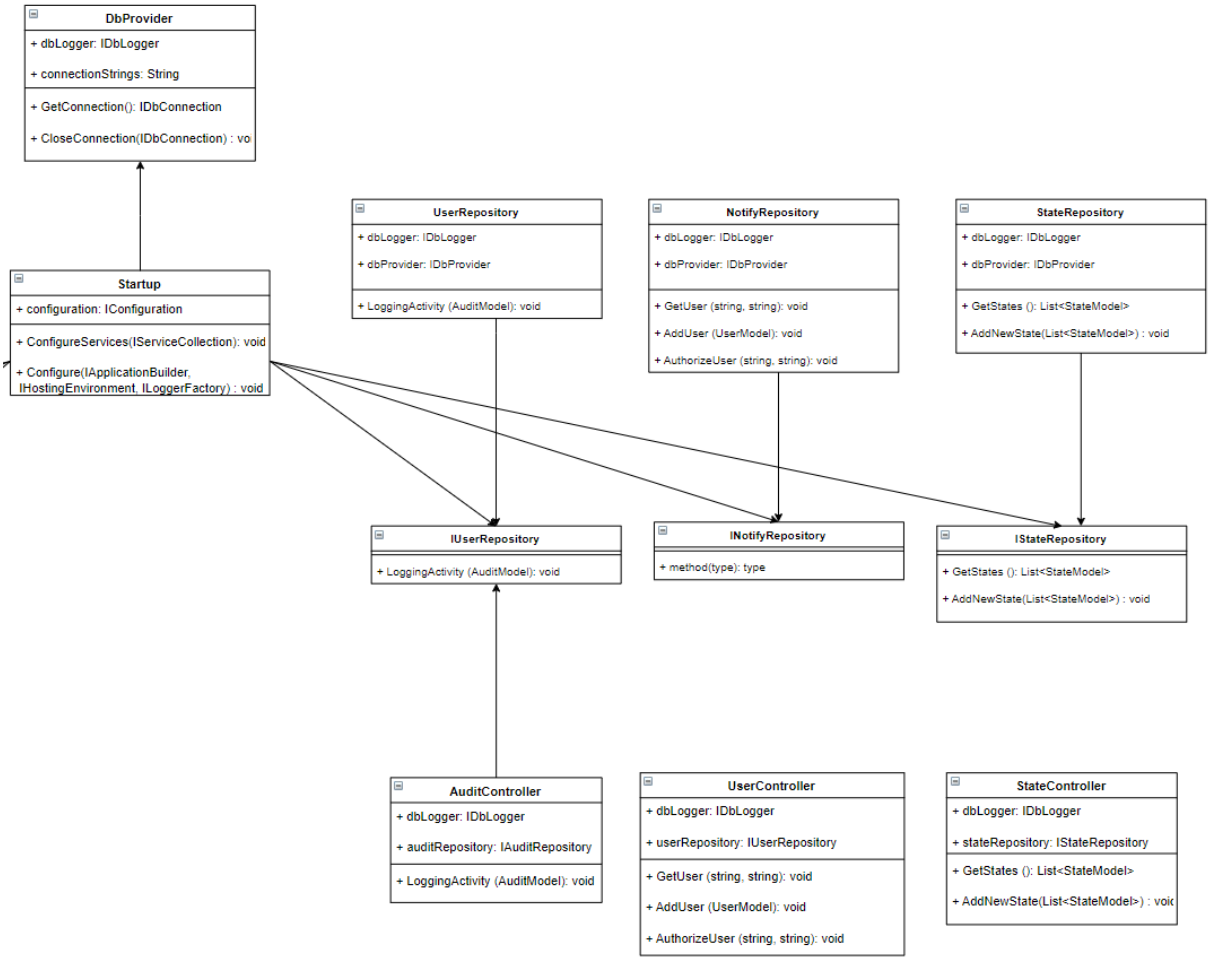


Рис. 3.3 Структурна схема класів програмного засобу

Класи, що використовуються в системі виявлення вторгнень наведено в табл. 3.9.

Таблиця 3.9

Список класів програмного застосунку

Клас	Опис
StateController	Клас для доступу до даних про стан веб-застосунку, що розміщені в базі даних
UserController	Клас для доступу до адміністраторів програмного засобу, що розміщені в базі даних

Продовження табл. 3.9

AuditController	Клас для доступу до даних про аудит, що розміщені в базі даних
SettingsController	Клас для доступу до даних налаштувань, задані адміністратором програмного засобу
NotifyController	Клас для відправки електронних листів на пошту адміністратора
AnalyzeController	Клас для аналізу HTTP-запитів на предмет SQL-ін’єкцій
ConnectionDbProvider	Клас для встановлення доступу до бази даних
StoreDataController	Клас для запису даних про SQL-ін’єкції до лог-файлу та бази даних

Список основних методів класів описаних в табл. 3.9 наведено в табл. 3.10 – 3.17.

Таблиця 3.10

Список основних методів класу StateController

Тип даних	Метод та опис
List	GetStates () Отримання стану веб-застосунку для перегляду адміністратором
Boolean	AddNewState (StateModel user) Додавання нового стану веб-застосунку для перегляду адміністратором до бази даних та лог-файлу

Таблиця 3.11

Список основних методів класу UserController

Тип даних	Метод та опис
Object	GetUser (string user, string password) Отримання даних адміністратора
Boolean	AddUser (UserModel userModel) Додавання нового адміністратора
Boolean	AuthorizeUser (string user, string password) Авторизування адмінстратора

Таблиця 3.12

Список основних методів класу AuditController

Тип даних	Метод та опис
Object	LoggingActivity (AuditModel auditModel) Запис дій адміністратора та веб-застосунку у базу даних

Таблиця 3.13

Список основних методів класу NotifyController

Тип даних	Метод та опис
Boolean	GetUserMail() Перевірка чи коректна електронна адреса та її існування
Boolean	SendMail(string text, string mail) Відправка повідомлення на пошту адміністратора
String	GenerateMessage(string subject, string message) Створення повідомлення

Таблиця 3.14

Список основних методів класу AnalyzeController

Тип даних	Метод та опис
List	ProcessRequests() Аналіз HTTP-запитів
Boolean	SaveEvents(list data) Збереження аналізованої інформації до бази даних системи

Таблиця 3.15

Список основних методів класу ConnectionToDB

Тип даних	Метод та опис
Object	GetConnection () Отримання доступу до бази даних
Boolean	CloseConnection (IDbConnection connection) Закриття сеансу з’єднання до бази даних

Таблиця 3.16

Список основних методів класу SettingController

Тип даних	Метод та опис
List	GetSettings () Отримання налаштувань програмного засобу, введені адміністратором веб-застосунку
Boolean	SetSettings () Додавання користувачем нових налаштувань програмного засобу виявлення SQL-ін’єкцій

Продовження табл. 3.16

Boolean	DeleteSetting (int id) Видалення певного налаштування адміністратора програмного засобу виявлення SQL-ін’єкцій
Boolean	DeleteAllSetting () Видалення усіх користувацьких налаштувань програмного засобу виявлення SQL-ін’єкцій
Boolean	EditSetting(list data) Редагування налаштувань програмного засобу виявлення SQL-ін’єкцій адміністратором веб-застосунку

Таблиця 3.17

Список основних методів класу StoreDataController

Тип даних	Метод та опис
Boolean	SaveToDb(object data) Збереження даних у базі даних
Boolean	SaveToLogFile(object data) Збереження даних у лог-файл

3.3. Визначення фізичної структури програмного забезпечення

Незалежно від способу зберігання посилань вони дають можливість будувати структури даних типу списків і дерев, мереж, таблиць зі змінною розмірністю. За допомогою структур даних моделюються відносини об'єктів реального світу, на їх основі будується алгоритм змістовної обробки цих даних. Спосіб представлення даних в пам'яті ЕОМ називається фізичною структурою даних [21].

Для визначення фізичної структури програмного засобу виявлення SQL-ін’єкцій було побудовано діаграми компонентів, які зображені на рис. 3.4 – 3.5 [10]. Повна діаграма компонентів програмного засобоу виявлення SQL-ін’єкцій зображена у дод. 1.

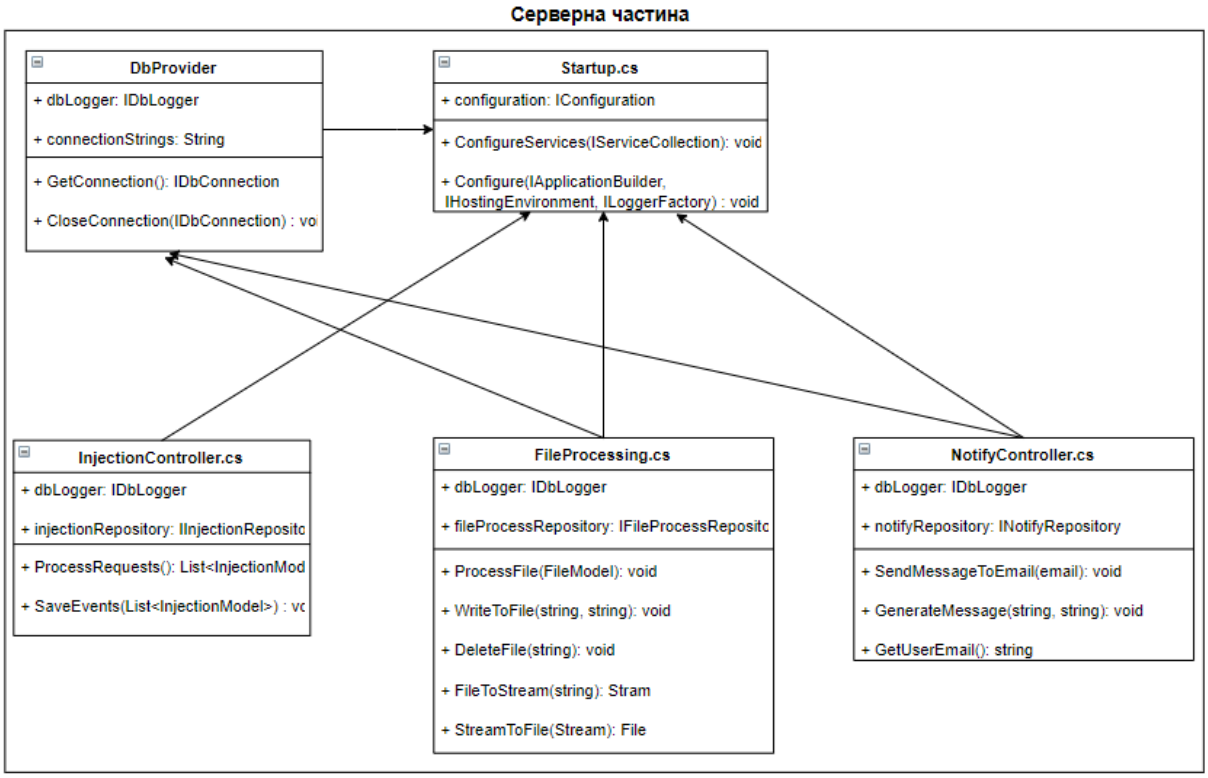


Рис. 3.4. Діаграма компонентів серверної частини програмного засобу виявлення SQL-ін’єкцій

На рис. 3.3 можна побачити структуру файлів програмного засобу виявлення SQL-ін’єкцій, а саме серверної частини, в яких реалізовано класи, що було описано у п. 3.2 [22]:

– Startup.cs – головний файл, який забезпечує роботу програмного засобу виявлення SQL-ін’єкцій;
– InjectionController.cs – містить у собі класи, які реалізують пошук, збір та аналіз SQL-ін’єкцій;
– DbProvider.cs – містить у собі клас, який реалізує доступ до інформації, що знаходиться у базі даних;

– FileProcessing.cs – містить у собі клас, який реалізує роботу з файлами: записи даних, читання даних;
– DbConnecction.cs – містить у собі клас, який реалізує доступ до бази даних;
– NotifyController.cs – містить у собі клас, який реалізує сповіщення адміністратора про SQL-ін’єкцію.

На рис. 3.5 можна побачити структуру файлів програмного засобу виявлення SQL-ін’єкцій, а саме клієнтської частини, в яких реалізовано класи, що було описано у п. 3.2:

– stateCtrl.js – файл з розширенням javaScript, який реалізує роботу клієнтської частини програмного засобу виявлення SQL-ін’єкцій, а саме роботою з даними про стан веб-застосунку;
– index.html – основна HTML-сторінка програмного засобу виявлення SQL-ін’єкцій;
– settings.html – додаткова HTML-сторінка для роботи з налаштуваннями програмного засобу виявлення SQL-ін’єкцій;

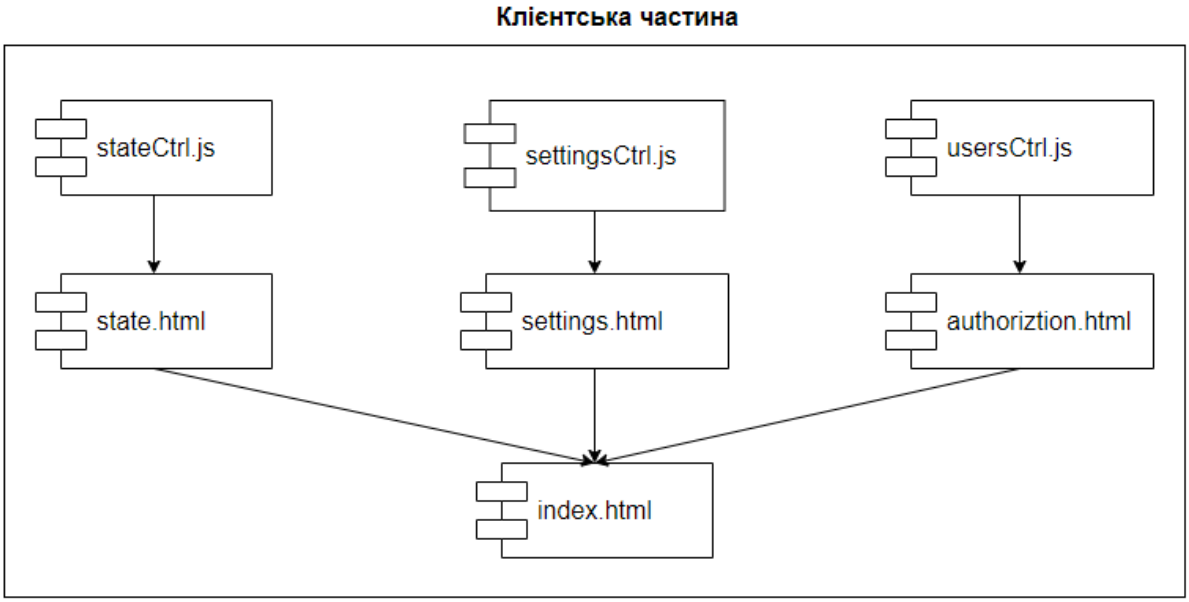


Рис. 3.5 Діаграма компонентів клієнтської частини програмного засобу виявлення SQL-ін’єкцій

– settingsCtrl.js – файл з розширенням javaScript, який реалізує роботу клієнтської частини програмного засобу виявлення SQL-ін’єкцій, а саме налаштування, видалення або додавання налаштувань адміністратором;
– authorization.html – додаткова HTML-сторінка для роботи з обліковими записами користувачів програмного засобу виявлення SQL-ін’єкцій;
– usersCtrl.js – файл з розширенням javaScript, який реалізує роботу клієнтської частини програмного засобу виявлення SQL-ін’єкцій, а саме авторизація, реєстрація та видалення адміністраторів.

3.4. Висновки

За результатами об’єктно-орієнтованого моделювання визначено варіанти використання, логічне, фізичне представлення даних та реалізовано програмне забезпечення для виявлення SQL-ін’єкцій у веб-застосунках.

Для проектування варіантів використання визначено дійових осіб та використано UML-діаграму варіантів використання.

Логічна структура містить набір функціонально-логічних модулів, що включають процедури і об'єкти, що представляють собою стандартні прототипи додатків баз даних та відображена у вигляді відповідної UML-діаграми

Для визначення фізичної структури програмного засобу виявлення SQL-ін’єкцій було побудовано діаграми компонентів.

Основні результати розділу опубліковано в роботі [10].

4. РОБОЧИЙ ПРОЕКТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЯВЛЕННЯ SQL-ІН'ЄКЦІЙ У ВЕБ-ЗАСТОСУНКАХ

4.1. Реалізування програмного забезпечення для виявлення SQL-ін'єкцій у веб-застосунках

Під час розробки програмного засобу виявлення SQL-ін'єкцій було прийнято деякі рішення, які стосувалися безпосередньо архітектури, технологій та шаблонів проектування програного забезпечення.

З існуючих видів застосунків, було обрано веб-застосування, тому було проведено попередній аналіз та затверджено вибір даного типу. Даний вибір повністю виправдав себе на практиці, підтвердженням цьому стали:

- масштабованість;
- модульність;
- зручне розподілення модулів для розробки між різними розробниками;
- вільний доступ з пристрою, з доступом до інтернету;
- не потребує оновлення системи, крім місця самого розгортання веб-застосунку;
- великій вибір допоміжних бібліотек та фреймфорків, що дало змогу вдало підібрати допоміжні інструменти для специфічних задач.

В якості мови програмування було обрано C#. Дана мова дозволяє реалізувати, як серверну так і клієнтську частини. Вона є широко вживаною, зручною у використанні, стрімко розвивається та підходить для будь-яких цілей.

Для спрощення розробки та кращої організації коду було обрано платформу ASP.NET Core 2.0. Мільйони розробників використовували ASP.NET 4.x для створення веб-програм. ASP.NET Core - це редизайн ASP.NET 4.x, з архітектурними змінами, що призводить до меншої, більш модульної структури [23].

Серверна частина реалізована за допомогою C#. Даний набір технологій є популярним та прослідкується швидкий розвиток. Першочергово, було сформовано структуру API, яку повинен був забезпечувати сервер, продумано всю маршрутизацію та потік даних, як на вхід так і на вихід [24].

Серверна частину можна розгорнути окремо від клієнтської частини, тому ці частини незалежні і мають свій простір для більшої масштабованості та вдосконалення.

ASP.NET Core надає наступні переваги [25]:

- єдиний підхід для створення веб-інтерфейсу та WebApi;
- архітектура адаптивна для проведення модульного тестування;
- Razor Pages робить сценарії, орієнтовані на кодування, більш простими та продуктивними;
- можливість розробки та запуску в Windows, MacOS та Linux.
- відкрите джерело та орієнтоване на розробників;
- інтеграція сучасних, клієнтських структур та робочих процесів розробки;
- обладнана система конфігурації;
- вбудована ін'єкція залежностей;
- легкий, високопродуктивний та модульний HTTP-запит;
- можливість хостингу в IIS, Nginx, Apache, Docker або самостійному хості у вашому власному процесі;
- параметри версії додатків при націлюванні на .NET Core;
- інструмент, що спрощує сучасну веб-розробку.

Для візуалізації інформації про вторгнення було використано мови розмітки HTML та мова AngularMaterail, яка дає можливість динамічно змінювати стани та види об’єктів та елементів [26].

Клієнтська частина реалізована на мові програмування JavaScript з використанням допоміжних фреймфорків, як:

- Angular;

Angular, підтримується компанією Google, - це платформа розробки програмного забезпечення з відкритим кодом, що використовується для побудови користувацьких інтерфейсів (зовнішній інтерфейс) [27].

Компоненти можна розглядати як маленькі частини інтерфейсу, незалежні один від одного. Дає можливість створювати прості додатки зі списком елементів і відповідним полем пошуку, щоб отримати елементи за словом. Блоки з перерахованими іменами, вікно пошуку та основний екран, де розміщені дві інші коробки, вважаються окремими компонентами в Angular [28].

4.2. Тестування програмного забезпечення для виявлення SQL-ін'єкцій у веб-застосунках

4.2.1. Аналіз якості програмного забезпечення

Для аналізу якості програмного засобу, необхідно визначити характеристики та підхарактеристики, яким воно повинно відповідати [29].

Існують чотири основні характеристики та їх підхарактеристики:

- функціональність – властивості, визначають спроможність ПЗ виконувати в заданому середовищі упорядковану послідовність дій для задоволення споживчих властивостей, замовлених користувачем, відповідно до вимог обробки і загальносистемних засобів. Атрибути функціональності ПЗ:
  - функціональна повнота – атрибут, який показує ступінь достатності основних функцій для вирішення завдань відповідно до призначення програмного забезпечення;
  - правильність – атрибут, який показує, як забезпечується досягнення правильних та погоджених результатів;
  - захищеність – атрибути, які вказують на можливість запобігати несанкціонованому доступу до програм і даних.

– зручність застосування – це множина атрибутів, що характеризують умови взаємодії користувача з ПЗ. Атрибути зручності застосування ПЗ:
  – зрозумілість – визначається, наскільки зрозумілі для розпізнавання логічні концепції ПЗ та умов їх застосування;
  – легкість навчання – визначається, наскільки доступні (легкі) для вивчення умови використання.
– ефективність – характеризується ступенем відповідності використовуваних ресурсів середовища до функціонування рівня якості.
  – час реакції – час відгуку, опрацювання, виконання функцій.

В табл. 4.1 наведено конфліктуючі між собою підхарактеристики з визначених.

Таблиця 4.1

Попарно конфліктуючі характеристики

Характеристика		Функціональність			Зручність застосування		Ефектив-ність
		Функціональна повнота	Правильність	Захищеність	Зрозумілість	Легкість навчання	Час реакції
Функціональність	Функціональна повнота	–	–	–	–	–	+
	Правильність	–	–	–	+	–	–
	Захищеність	–	–	–	–	–	–

Продовження табл. 4.1

Зручність	Зрозумілість	–	+	–	–	–	–
	Легкість навчання	–	–	–	–	–	+
Ефективність	Час реакції	+	–	–	–	+	–

В табл. 4.2 наведено метрики та вагові коефіцієнти для характеристик.

Таблиця 4.2

Метрики та вагові коефіцієнти характеристик

Характеристика	Підхарактеристика	Метрика	Вага
Функціональність	Функціональна повнота	Число реалізованих функцій від загального числа заявлених функцій	0.9
	Правильність	Число результатів від загального числа очікуваних результатів	0.8
	Захищеність	Число захищеності програмного забезпечення	0.8
Зручність застосування	Зрозумілість	Число зрозумілості використання програмного забезпечення	0.6

Продовження табл. 4.2

Зручність застосування	Легкість навчання	Число легкості навчання програмного забезпечення	0.6
Ефективність	Час реакції	Число часу відгуку програми на дії користувача	0.8

Детальніше взаємозв’язок характеристик зовнішньої та експлуатаційної якості наведено на рис. 4.1.

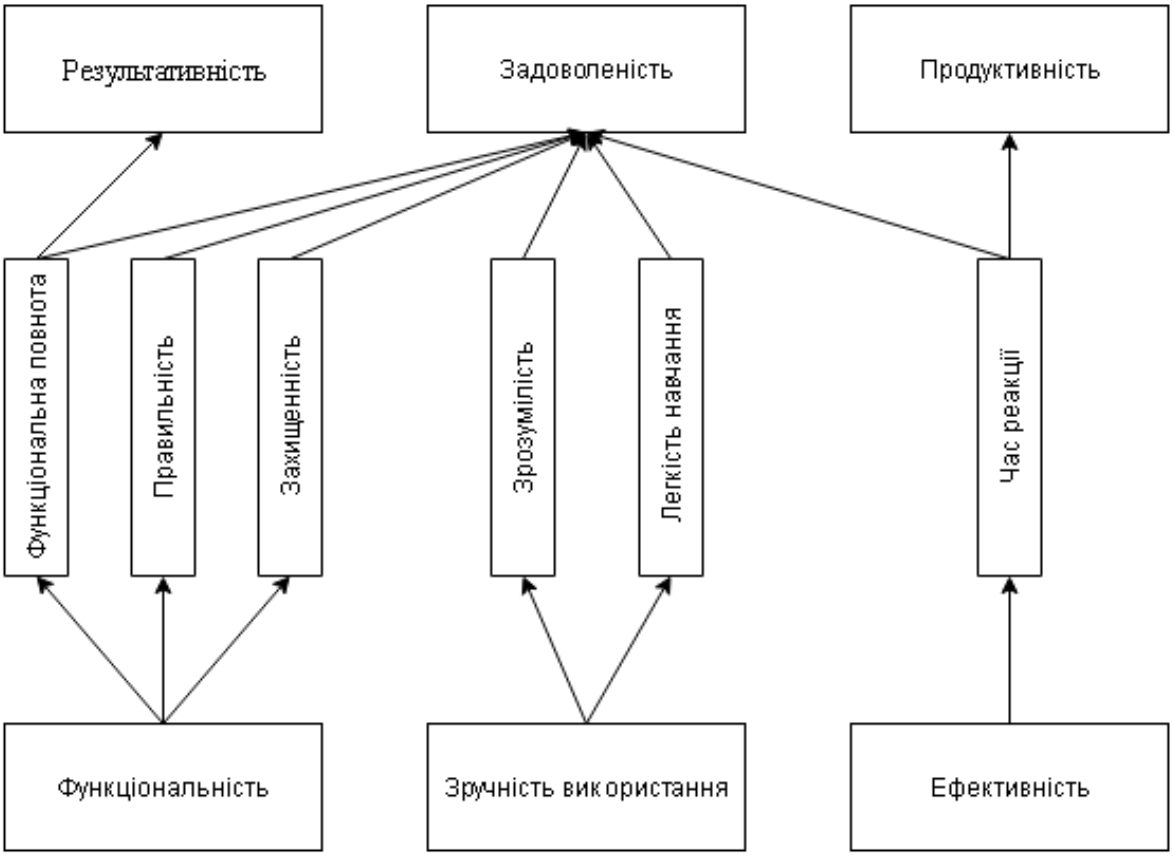


Рис. 4.1. Взаємозв’язок зовнішньої та експлуатаційної якості

Визначено наступні характеристики експлуатаційної якості:

– результативність – використовується, тому що є важливим оцінка якості виконання програмою своїх безпосередніх функцій та

отримання результату, що за своєю точністю задовольнить користувача;

– продуктивність – ступінь, в якій витрачаються ресурси на досягнення користувачем заданої ефективності вирішення завдань у встановленому контексті використання програмного забезпечення;
– задоволеність – визначає ступінь, в якій користувач задоволений програмним забезпеченням в межах його використання.

4.2.2. Опис процесів тестування

Тестування програмного забезпечення – процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснений шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів. Таким чином тестування є своєрідним методом контролю якості тих характеристик ПЗ, що проявляються в процесі його роботи [30]. Також тестування надає наочну оцінку системи для того, щоб знайти відмінності між тим, яким програмне забезпечення повинно бути і якою воно є. Тестування поділяється на велику кількість різних видів. Для перевірки невідповідності між реальною поведінкою функцій, що були реалізовані в рамках ПЗ, і очікуваною поведінкою, визначеною вимогами до ПЗ використовується функціональне тестування [31].

Під час тестування програмного засобу виявлення SQL-ін’єкцій необхідно перевірити функціональну відповідність наступних функцій [32]:

– збереження інформації про SQL-ін’єкції до лог-файлу;
– запис інформації про SQL-ін’єкції до бази даних;
– надсилання інформації про SQL-ін’єкції на адміністратора.

Застосування буде вважатися таким, що пройшло тестування, якщо

– результати виконання функцій будуть відповідати очікуваним;

– при введені некоректних даних застосування буде фіксувати це та повідомляти про помилку;
– дані у сформованих файлах та повідомленнях коректні та відповідають даним, введеним в додаток.

4.2.3. Опис контрольного прикладу

При тестуванні програмного забезпечення було перевірено функціональність розроблюваного програмного засобу відповідно до вимог у п. 4.2. В табл. 4.3 - 4.5 наведено інформацію про тестування основних варіантів використання та їх результати зображені на рис. 4.2 та 4.3 [10, 33].

Таблиця 4.3

Перевірка запису інформації про SQL-ін’єкцію у лог-файл

Мета тесту	Перевірка можливості запису інформації про SQL-ін’єкції у лог-файл
Початковий стан	Відкритий програмне забезпечення виявлення SQL-ін’єкцій
Вхідні дані	Дані про зловмисника, який намагається виконаим впровадження SQL-коду: «Код події», «Текст події», «Користувач», «Дата», «Час», «Шлях запиту»
Схема проведення тесту	Перейти на веб-застосунок та виконати HTTP-запит з впровадженням SQL-коду
Очікуваний результат	Дані записані до лог-файлу
Стан програмного продукту після проведення випробувань	Дані записані до лог-файлу

Таблиця 4.4

Перевірка запису інформації про SQL-ін’єкцію до бази даних

Мета тесту	Перевірка запису інформації про SQL-ін’єкцію до бази даних
Початковий стан	Відкритий програмне забезпечення виявлення SQL-ін’єкцій
Вхідні дані	Дані про зловмисника, який намагається виконаим впровадження SQL-коду: «Код події», «Текст події», «Користувач», «Дата», «Час», «Шлях запиту»
Схема проведення тесту	Перейти на веб-застосунок та виконати HTTP-запит з впровадженням SQL-коду
Очікуваний результат	Дані записані до бази даних
Стан програмного продукту після проведення випробувань	Дані записані до бази даних

Після перевірки контрольного прикладу «Перевірка запису інформації про SQL-ін’єкцію до бази даних» при роботі з програмним засобом виявлення SQL-ін’єкцій отримано такий результат: інформація про SQL-ін’єкції записана до бази даних (див. рис. 4.2).

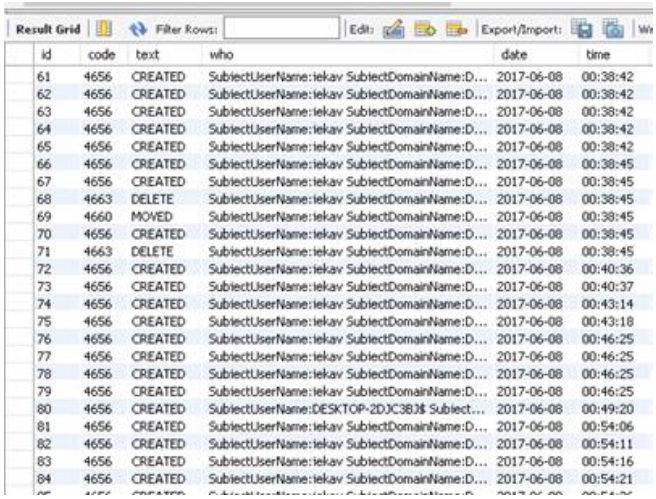


Рис. 4.2. Результат запису даних до бази даних

Таблиця 4.5

Перевірка відправки повідомлення на пошту користувача

Мета тесту	Перевірка можливості відправки повідомлення на пошту адміністратора
Початковий стан	Відкритий програмне забезпечення виявлення SQL-ін’єкцій
Вхідні дані	Дані про зловмисника, який намагається виконаим впровадження SQL-коду: «Код події», «Текст події», «Користувач», «Дата», «Час», «Шлях запиту»
Схема проведення тесту	Перейти на веб-застосунок та виконати HTTP-запит з впровадженням SQL-коду
Очікуваний результат	Повідомлення успішно надіслано на пошту адміністратора
Стан програмного продукту після проведення випробувань	Повідомлення успішно надіслано на пошту адміністратора

Після перевірки контрольного прикладу «Перевірка можливості відправки повідомлення на пошту адміністратора» при роботі з програмним засобом виявлення SQL-ін’єкцій було отримано такий результат: повідомлення успішно надіслано на пошту користувача, що можна побачити на рис. 4.3.

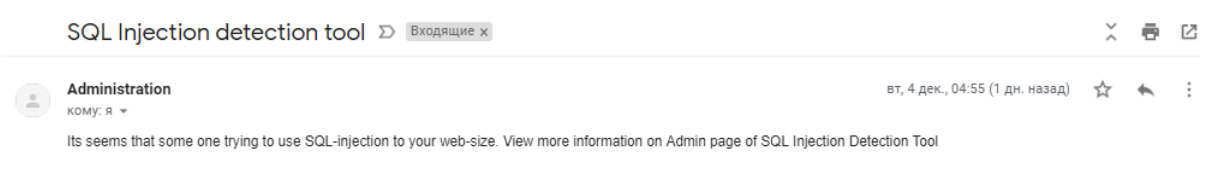


Рис. 4.3. Результат відправки повідомлення на пошту

4.3. Використання програмного забезпечення для виявлення SQL-ін’єкцій у веб-застосунках

4.3.1. Системні вимоги до програмного забезпечення

Програмний засіб виявлення SQL-ін’єкцій у веб-застосунках побудований на основі клієнт-серверної архітектури [34].

Клієнтська частина є web-інтерфейсом, для її роботи достатньо встановленого web-браузера та наступних бібліотек: AngularJS, AngularMaterial, Kendo. Для нормальної роботи програмного забезпечення достатньо встановити один з наступних браузерів: Google Chrome, Mozilla Firefox, Internet Explorer 9+ та в Safari 6+. Програмне забезпечення повинно працювати під управлінням операційних систем Windows XP з пакетом оновлень 2+, Windows Vista, Windows 7, Windows 8, Windows 10.

Для нормальної роботи серверної частини необхідно встановити операційну систему сімейства Windows та наступне програмне забезпечення: Python2.7, PythonDjango, MySQL Server, MySQL 5.0 або новіша.

4.3.2. Апаратні вимоги до програмного забезпечення

Вимоги до апаратної конфігурації клієнта не пред’являються, достатнім є виконання системних вимог [35].

Мінімальні апаратні вимоги до серверу, на якому буде розміщуватись серверна частина програмного засобу наступні:

- процесор на основі команд x86, Intel Pentium 4, Pentium D, Core, AMD Athlon, Phenom, або інший CISC-процесор чи гібридний процесор;
- об‘єм ОЗП: 512Мб;
- частота процесору: 1 ГГц;
- активне підключення до глобальної мережі.

4.3.3. Розгортання програмного забезпечення

Перед розгортанням програмного забезпечення на сервері, необхідно переконатися, що сервер відповідає системним та апаратним вимогам наведеним у пунктах 4.3.1 та 4.3.2.

До складу файлів, необхідних для встановлення програмного забезпечення входять наступні [36]:

- Sql.Injction.Tool.Back – серверна частина, як модуль виявлення вторгнень;
- Sql.Injction.Tool.Front – клієнтська частина, як модуль відображення даних;
- dbsit.sql – база даних.

Дії, необхідні для розгортання програмного забезпечення на сервері, наведено далі:

- Дія 1. Створити нову базу даних, та експортувати до неї базу даних з файлу dbsit.sql.
- Дія 2. Скопіювати модуль Sql.Injction.Tool.Back на робочу машину та розгорнути веб-сервер;
- Дія 3. Скопіювати модуль Sql.Injction.Tool.Front на робочу машину та розгорнути веб-сервер;
- Дія 3. Вести параметри для моніторингу сервера.

В процесі розгортання програмного забезпечення можуть виникнути помилки, їх причини та способи усунення наведено нижче:

- Застосування працює, але замість кириличних символів відображаються символи «?». Причина – неправильне кодування бази даних, встановіть кодування utf8.

4.4. Висновки

Для розробки програмного засобу виявлення SQL-ін’єкцій прийнято рішення, які стосувалися безпосередньо архітектури, технологій та

шаблонів проектування програного забезпечення. Зокрема в якості мови програмування для реалізації серверної частини було обрано мову C# та Angular 6 для реалізації клієнтської частини.

Визначено характеристики та метрики якості метрики тестування. При тестуванні програмного забезпечення було перевірено функціональність розроблюваного програмного засобу відповідно до вимог та проведено тестування основних варіантів використання програмного забезпечення.

Визначено системні та апаратні вимоги до клієнтської та серверної частини програмного забезпечення. Описано кроки успішного встановлення програмного забезпечення.

5. СТАРТАП-ПРОЕКТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЯВЛЕННЯ SQL-ІН'ЄКЦІЙ У ВЕБ-ЗАСТОСУНКАХ

5.1. Опис ідеї проекту

SQL-ін'єкція - це ін'єкційна атака, в якій зловмисник може виконувати шкідливі твердження SQL (також зазвичай називають шкідливим корисним завантаженням), які керують сервером бази даних веб-додатків (також зазвичай називають реляційною системою керування базами даних). Оскільки вразливість SQL-ін'єкції може вплинути на будь-який веб-сайт або веб-додаток, що використовує базу даних SQL, вразливість є однією з найстаріших, найбільш поширених та найбільш небезпечних вразливостей веб-програм [37].

Використовуючи вразливість SQL-ін'єкції, з урахуванням правильних обставин, зловмисник може використати його, щоб обійти механізми автентифікації та авторизації веб-додатків та отримувати вміст всієї бази даних. SQL-ін'єкції також може використовуватися для додавання, зміни та видалення записів у базі даних, що впливає на цілісність даних [38].

У такому випадку SQL-ін'єкції може надати зловмиснику несанкціонований доступ до конфіденційних даних, включаючи дані клієнта, персональну ідентифікаційну інформацію, комерційну таємницю, інтелектуальну власність та іншу конфіденційну інформацію.

SQL - це мова програмування, призначений для управління даними, тому SQL можна використовувати для доступу, модифікації та видалення даних. Крім того, в окремих випадках система також може запускати команди операційної системи з SQL-оператора.

З огляду на наведене вище, легше зрозуміти, наскільки прибутковою може бути успішна атака SQL-ін'єкції для атакуючого.

Зловмисник може використовувати SQL-ін'єкцію, щоб обійти автентифікацію або навіть видати себе за конкретних користувачів.

Однією з основних функцій SQL є вибір даних на основі запиту і виведення результату цього запиту. Вразливість SQL Injection може дозволити повне розкриття даних, що знаходяться на сервері баз даних.

Оскільки веб-застосунки використовують SQL для зміни даних у базі даних, зловмисник може використовувати SQL- ін’єкцію для зміни даних, що зберігаються в базі даних. Змінення даних впливає на цілісність даних і може спричинити заперечення, наприклад, такі питання, як скасування транзакцій, зміна балансу та інших записів.

SQL використовується для видалення записів з бази даних. Зловмисник може використовувати вразливість SQL-ін’єкцію для видалення даних з бази даних. Навіть якщо застосовується відповідна стратегія резервного копіювання, видалення даних може вплинути на доступність програми до відновлення бази даних.

Деякі сервери баз даних налаштовані (навмисне або інше), щоб дозволити довільне виконання команд операційної системи на сервері баз даних. З огляду на правильні умови, зловмисник може використовувати SQL- ін’єкцію як початковий вектор при нападі внутрішньої мережі, яка знаходиться за брандмауером.

На рис. 5.1 зображено дерево проблем, побудоване на основі виявлених проблем до означеної теми [39].

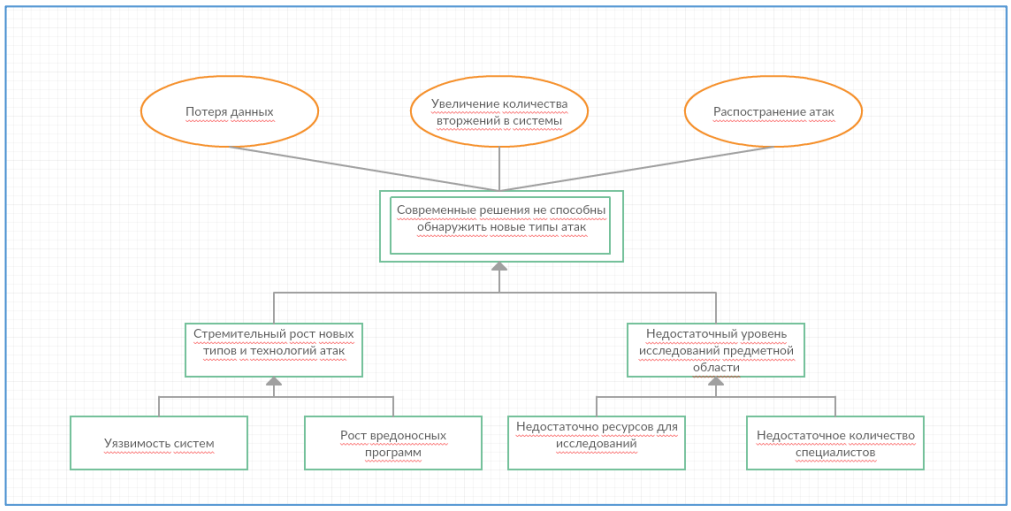


Рис. 5.1 Дерево проблем

Виходячи з дерева проблем, яке зображено на рис. 5.1 основною проблемою є неспроможність сучасних програмних засобів виявлення SQL-ін’єкцій знаходити та запобігати новим технологіям та типам атак.

5.2. Аналіз ринкових можливостей запуску стартап-проекту

Зацікавлена сторона проекту (ЗС) - особа, група або організація, яка може впливати на проект, або на яку можуть вплинути результати проекту або окремі завдання проекту [40].

Як можна зробити висновок з визначення, зацікавлена сторона може бути як зовнішньої, так і внутрішньої по відношенню до проекту.

Отже, у зацікавленої сторони є інтереси до проекту, які можуть бути порушені як позитивно, так і негативно в ході виконання або в результаті завершення проекту. Крім цього, різні зацікавлені сторони можуть мати суперечать очікування.

Ось і виходить, що якщо хтось має якісь очікування від проекту і може впливати на його хід, а керівник проекту про це не знає, то дії зацікавленої сторони можуть стати для нього неприємним сюрпризом [41].

Інформація про очікування зацікавлених сторін може в першу чергу вплинути на такі аспекти управління проектом, як збір і аналіз вимог до результатів проекту та аналіз ризиків проекту.

Мені відомі три ключові підходи до виявлення ЗС:

- Колесо зацікавлених сторін проекту
- Номінування зацікавлених сторін
- Вивчення документації по проекту

В основі першого підходу лежить використання класифікатора зацікавлених сторін. Природно, якщо б класифікатор ЗС був розроблений з урахуванням галузевої специфіки проекту, то його корисність була б великою. Але при відсутності галузевого класифікатора можна

використовувати універсальний, запозичивши його в одній з методологій з управління проектами [42].

Отже, в підході до управління проектом магістерської дисертації розглядається наступний список зовнішніх зацікавлених сторін:

– Компанії, які займаються розробкою веб-застосунків, або використовують системи у вигляді веб-застосунків;
– Компанії, які забезпечують серверне та комп’ютерне обладнання;
– Акціонери;
– Держава;
– Соціум;
– Інші ЗС.

У табл. 5.1 зображено аналіз зовнішніх зацікавлених сторін [43].

Таблиця 5.1

Аналіз зовнішніх зацікавлених сторін

Фактори	Можливості	Влада	Вплив		
			Влада	Інтерес	Сума
Компанії, які займаються розробкою веб-застосунків	Дають ідеї вдосконалити програмний продукт (відгуки, опитування і т.д.)	Забезпечують дохід	6	8	48
Компанії, які забезпечують серверне та комп’ютерне обладнання	Забезпечують потрібним програмним забезпеченням та обладнанням. Післяпродажне обслуговування	Залежність від якості наданої продукції	5	9	45

Продовження табл. 5.1

Акціонери	Дають згоду або не згоду на нові можливості або пропозиції	Рішення питань по управлінню засобами	7	10	70
Держава	Контроль податків. Регулювання діяльності. Введення обмежень	Обмеження у рамках закону	4	3	12
Соціум	Розширення клієнтської бази	Диктує попит	4	2	8

Отже, в підході до управління проектом магістерської дисертації розглядається наступний список внутрішніх зацікавлених сторін:

- Топ-менеджери;
- Менеджери;
- Розробники;
- Бізнес-аналітики;
- Технічний персонал .

У табл. 2 зображено аналіз зовнішніх зацікавлених сторін.

Таблиця 5.2

Аналіз внутрішніх зацікавлених сторін

Фактори	Сила	Слабкість	Вплив		
			Влада	Інтерес	Сума
Топ менеджери	Приймають головні рішення	Відповідальні за прийняття рішень	9	10	90

Продовження табл. 5.2

Менеджери	Організація продуктивності роботи персонала	Зобов’язані вирішувати велику кількість задач в короткий проміжок часу	4	5	20
Розробник	Забезпечення якості наданого продукта	Недосвідченість	6	1	6
Бізнес-аналітики	Оцінюють фінансовий та операційні аспекти роботи компанії та визначають перспективи її розвитку	Помилки при аналізі діяльності і оцінці перспектив компанії	5	5	25
Технічний персонал	Забезпечення комфортних умов роботи персоналу	Не суттєво впливають на роботу компанії	1	4	4

Споживачі:

- ранні клієнти: невеликі IT компанії;
- IT компанії та компанії-замовники програмного забезпечення.

Проблема:

- Сучасті методи та рограмні засоби нездатні виявити вторгнення до системи.

Рішення:

- Програмний засіб виявлення SQL-ін’єкцій у веб-застосунках.

Унікальна ціннісна пропозиція:

- гнучкі підписки для IT компаній так компаній-замовників програмних продуктів;
- можливість легкої інтеграції в існуючі системи і індивідуального налаштування;

– забезпечення технічною підтримки в робочі години.

Потоки доходів:

– продаж довгострокових підписок з усіма функціями для IT компаній;
– продаж довгострокових підписок з обраними функціями для IT компаній;
– продаж розширеної технічної підтримки;
– продаж довгострокової підписки з основними функціями для компаній-замовників програмного забезпечення;
– продаж короткострокової підписки з основними функціями для компаній-замовників програмного забезпечення.

Структура витрат:

– витрати на розроблення, підтримку та вдосконалення програмних продуктів;
– витрати на надання послуг технічної підтримки;
– витрати на оплату праці;
– витрати на оренду офісного приміщення, опалення, освітлення, водопостачання;
– адміністративні витрати;
– витрати на рекламу та дослідження ринку.

Також в канву бізнес-моделі включаються структурні блоки, які ще не були розглянуті. Це прихована перевага (перевага, яку не можливо скопіювати або купити), ключові метрики (основні показники, що вимірюються) та канали (шляхи до користувачів).

В якості каналів контакту з клієнтами пропонується використовувати наступні шляхи: тематичні ресурси та спільноти, профільні видання, прямі контакти для продажів.

Прихованою перевагою системи виступає використання синтезу методів для забезпечення кращого аналізу даних та побудови дерева рішень.

Ключовими метриками є наступні: кількість проданих підписок різних типів, кількість продажів індивідуальних налаштувань системи, кількість запитів в службу технічної підтримки.

5.3. Розроблення маркетингової програми стартап-проекту

Сумарний дохід складається як сума доходів від продаж товарів та супутніх послуг для кожного сегменту споживачів [44].

Для сегменту клієнтів пропонується продавати наступні товари та надавати наступні послуги:

- довгострокові підписки з усіма функціями системи;
- довгострокові підписки з обраними функціями системи;
- розширена технічна підтримка.

Під довготривалими підписками з обраними функціями системи мається на увазі, що клієнт може обрати один з запропонованих строків підписки та обрати одну або декілька функцій системи. Відповідно до цих параметрів і буде розраховуватись вартість підписки.

Розширення технічна підтримка пропонується як послуга на термін дії підписки і включає в себе надання консультацій, налаштування системи та виправлення помилок в робочі години усі дні, крім вихідних та свят [45].

Для сегменту клієнтів, що є замовниками програмного забезпечення, пропонується продавати наступні товари:

- довгострокову підписку з основними функціями системи;
- короткострокову підписку з основними функціями системи.

Різниця між довгостроковою та короткостроковою підпискою полягає в тому, що довгострокова підписка дається мінімум на півроку та є актуальною для компаній посередників між замовниками та IT компаніями.

В свою чергу, короткострокова підписка буде актуальною для замовників фізичних осіб, які рідко замовляють програмне забезпечення.

Розрахований дохід по місяцям протягом першого року наведено в табл. 5.3. Всі суми наведено в гривнях [46].

Таблиця 5.3

Доходи

	Довготривалі підписки	Усі функції системи	Обрані або основні функції системи	Розширена технічна підтримка	Всього
	1000	500	500	500	3000
2	3000	1500	1000	1500	8500
3	5000	3000	3000	2500	15500
4	10000	5000	5000	5000	30000
5	10000	5000	5000	5000	30000
6	12000	6000	4000	5500	32500
7	12000	6000	4000	5500	32500
8	12000	6000	4000	5500	32500
9	15000	10000	7000	9000	48000
10	15000	10000	7000	9000	48000
11	25000	10000	7000	9000	58000
12	27000	11000	7000	10000	63000
Сумарно за рік:					401500

Сукупність загальних витрат складають наступні витрати:

– витрати на розроблення, підтримку та вдосконалення програмних продуктів (утримання робочих місць, придбання необхідних інструментів та програмних засобів);

– витрати на надання послуг технічної підтримки (утримання робочих місць, придбання необхідних інструментів та програмних засобів);
– витрати на оплату праці (заробітна плата та інші виплати працівникам);
– витрати на оренду офісного приміщення, опалення, освітлення, водопостачання;
– адміністративні витрати (витрати на зв’язок, податки та збори, плата за послуги банків тощо);
– витрати на рекламу та дослідження ринку.

Витрати по місяцям протягом першого року наведено в табл. 5.4. Всі суми наведено в гривнях [47].

Таблиця 5.4

Витрати

	Технічна підтримка	Оплата праці	Комунальні та адміністративні	Реклама	Всього
1	5000	15000	5000	10000	35000
2	500	15000	5000	5000	25500
3	500	15000	5000	5000	25500
4	500	15000	5000	3000	23500
5	500	15000	5000	3000	23500
6	500	15000	5000	3000	23500
7	500	15000	5000	3000	23500
8	500	10000	5000	3000	18500
9	500	10000	5000	3000	18500

Продовження табл. 5.4

10	500	10000	5000	3000	18500
11	500	10000	5000	3000	18500
12	500	10000	5000	3000	18500
Сумарно за рік:					272500

Розрахуємо маржинальний прибуток за перший рік за формулою (5.1).

$$\text{Маржинальний прибуток} = \text{Дохід} - \text{Витрати} \qquad (5.1)$$

За формулою (5.1) маржинальний прибуток за перший рік складе 401500-(30000+272500) = 99000 гривень.

5.4. Розроблення ринкової стратегії проекту

Унікальна пропозиція вартості (UVP), пропозиція вартості - це коротка заява, яка охоплює переваги наданих послуг, а також того, як продукт відрізняєте себе від ваших конкурентів [48].

Цінова пропозиція повинна виконати чотири речі:

- Повинно зафіксувати увагу відвідувача.
- Повинно бути легко зрозумілим. Ділові стосунки коштують менше, ніж вартість штатного працівника, забезпечуючи десятирічний досвід.
- Повинен відрізняти продукт від ваших конкурентів в Інтернеті. Якщо список пропозицій стосовно вартості є подібним до конкурентів, потрібно зосередитися на тому, на якому вони не зосереджені.

– має бути достатньо привабливим, щоб дійсно вплинути на рішення про покупку відвідувача.

Ціннісна пропозиція описує ті переваги, які надають споживачу наші товари та послуги і які вирішують проблеми споживачів. Відповідно, унікальною буде така пропозиція, яка чітко відділяє товар від конкурентів і дає зрозуміти, чому необхідно придбати саме наш товар [49].

Для кожного сегменту споживачів зазвичай пропонується виділяти окрему ціннісну пропозицію.

Ціннісною пропозицією для сегменту клієнтів, які використовують веб-застосунки є система виявлення вторгнень у систему, що містить всі функції, а саме робить точний аналіз системи в залежності від впливу різних факторів та ризиків [50].

Для сегменту клієнтів, що є замовниками програмного забезпечення ціннісною пропозицією є спрощений варіант системи, що відображає стан системи та надає дані для моніторингу.

Для обох сегментів можна запропонувати звичайну або розширену технічну підтримку, звичайна технічна підтримка може здійснюватися один раз на добу, а розширена, наприклад, кожного дня в робочі години, окрім вихідних днів та державних свят.

З розширеною технічною підтримкою можливі збої та проблеми в роботі системи повинні бути усунені за дві години, як була надіслана заявка від клієнта.

Також клієнтів можна зацікавити регулярними оновленнями програми та стабільними версіями.

5.5. Висновки

Завдяки аналізу та опису проблема означеної теми було побудоване дерево проблем, завдяки якої було виявлено, що основною проблемою є неспроможність сучасних програмних засобів виявлення SQL-ін’єкцій

знаходити та запобігати новим технологіям та типам атак, тому на основі цього стало можливим визначення ринкової комерціалізації проекту.

Були визначені зацікавлені сторони та їх інтереси до проекту, які можуть бути порушені як позитивно, так і негативно в ході виконання або в результаті завершення проекту. Крім цього, різні зацікавлені сторони можуть суперечити очікуванням.

Для сегменту клієнтів, що є замовниками програмного забезпечення, пропонується продавати наступні товари:

– довгострокову підписку з основними функціями системи;
– короткострокову підписку з основними функціями системи.

Для сегменту клієнтів, що є замовниками програмного забезпечення ціннісною пропозицією є спрощений варіант системи, що відображає стан системи та надає дані для моніторингу.

Для обох сегментів можна запропонувати звичайну або розширену технічну підтримку, звичайна технічна підтримка може здійснюватися один раз на добу, а розширена, наприклад, кожного дня в робочі години, окрім вихідних днів та державних свят.

ВИСНОВКИ

У магістерській дистертації вирішено актуальне завдання, зокрема, розроблено програмне забезпечення для виявлення SQL-ін’єкцій у веб-застосунках.

При цьому отримано такі результати:

1. Проаналізовано програмні засоби виявлення SQL-ін’єкцій стосовно своєчасності оповіщення про них. Внаслідок такого аналізу показано необхідність створення відповідного програмного забезпечення та виокремлено його місце на графічній схемі. На основі проаналізованої інформації сформовано функціональні та нефункціональні вимоги та поставлено цілі, які необхідно досягнути в процесі розробки.

2. Побудовано концептуальну модель програмного забезпечення виявлення SQL-ін’єкцій у веб-застосунках, використання якої дозволяє формалізувати його роботу на рівні функцій, процесів і потоків даних, а також сформувати та обґрунтувати варіанти використання програмного забезпечення.

3. За результатами концептуального моделювання побудовано об’єктно-орієнтовану модель програмного забезпечення на основі формалізування його роботи на рівні функцій, процесів і потоків даних, використання якої дозволяє надати нову якість програмному забезпеченню при створенні або вдосконаленні, зокрема, забезпечити їх функціональну придатність до виявлення SQL-ін’єкцій у веб-застосунках.

4. Визначено характеристики та метрики якості програмного забезпечення і проведено функціональне тестування. Наведено контрольні приклади для проведення тестування основних варіантів використання. Визначено системні та апаратні вимоги до клієнтської та серверної частини розроблюваного програмного засобу. Описано кроки, які необхідно виконати для успішного встановлення програмного засобу. Також наведено інструкцію з користування програмним забезпеченням для адміністратора.

5. Визначено ринкові перспективи проекту, графік та принципи організації виробництва, фінансовий аналіз та аналіз ризиків і заходи з просування пропозиції для інвесторів. Узагальнено етапи розроблення стартап-проекту для розроблення та виведення стартап-проекту на ринок передбачає здійснення низки кроків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фаззинг [Електронний ресурс]. – Режим доступу: https://goo.gl/3pyJSN.

2. LDAP (Lightweight Directory Access Protocol) [Електронний ресурс]. – Режим доступу: https://goo.gl/wfGKss.

3. NoSQL [Електронний ресурс]. – Режим доступу: https://goo.gl/pYS5oz.

4. XPath [Електронний ресурс]. – Режим доступу: https://goo.gl/UCDbu6.

5. Cross-site scripting [Електронний ресурс]. – Режим доступу: https://goo.gl/Ja1Y14.

6. Pietraszek T. Defending against Injection Attacks through Context-Sensitive String evaluation / T. Pietraszek, C. V. Berghe // Recent Advances in Intrusion Detection. – 2013. – Vol. 3858. - pp. 124- 145.

7. Code injection [Електронний ресурс]. – Режим доступу: https://goo.gl/4i2HEE.

8. Top 10 A1-Injection [Електронний ресурс]. – Режим доступу: https://goo.gl/comHQX.

9. SQL Injection Prevention Cheat Sheet [Електронний ресурс]. – Режим доступу: https://goo.gl/N1NHtW.

10. Васін Є.В. Програмний засіб виявляння SQL-ін’єкцій у веб-застосунках / Є.В. Васін, В.В. Цуркан // XI наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК-2018-2), 2018. – 4 с.

11. Gould C. JDBC Checker:A Static Analysis Tool for SQL/JDBC Applications / C. Gould, Z. Su, P. Devanbu // In Proceedings of the 26th International Conference on Software Engineering (ICSE 04) Formal Demos, 2004. – pp 697-698.

12. Wassermann G. Static Checking of Dynamically Generated Queries in Database Applications / Wassermann, G; Gould, C; Su, Z, et al. // ACM Transactions on Software Engineering and Methodology. – 2017. – Vol. 16, Iss. 4. – 10 p.

13. IDEF0 [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/IDEF0.

14. Нотация IDEF0 [Електронний ресурс]. – Режим доступу: https://goo.gl/bH5XYw.

15. IDEF3 [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/IDEF3.

16. Основи IDEF3 [Електронний ресурс]. – Режим доступу до ресурсу: http://citforum.ck.ua/cfin/idef/idef3.shtml.

17. DFD [Електронний ресурс]. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/DFD.

18. Диаграмма потоков данных (DFD). Графический язык диаграммы. Примеры [Електронний ресурс]. – Режим доступу до ресурсу: http://e-educ.ru/bd14.html.

19. Діаграма варіантів використання (use case diagram) [Електронний ресурс]. – Режим доступу до ресурсу: http://5fan.ru/wievjob.php?id=21296.

20. Логическая структура программного обеспечения [Електронний ресурс]. – Режим доступу до ресурсу: https://goo.gl/R4KIrC.

21. Отношения классов – от UML к коду [Електронний ресурс]. – Режим доступу до ресурсу: https://goo.gl/9qELSU.

22. Физическая структура программного обеспечения [Електронний ресурс]. – Режим доступу: https://goo.gl/5gEVc5.

23. Upgrading to Angular 6 [Електронний ресурс]. – Режим доступу: https://goo.gl/TNSM2k.

24. NET Core [Електронний ресурс]. – Режим доступу: https://goo.gl/Ju65Hx.

25. Руководство по ASP.NET Core 2 [Електронний ресурс]. – Режим доступу: https://metanit.com/sharp/aspnet5/.

26. Top 6 reasons to use Angular 5 framework for your project — [Електронний ресурс]. – Режим доступу: https://goo.gl/AKvHj7

27. Контрольный пример [Електронний ресурс]. – Режим доступу: https://goo.gl/GuZSXM.

28. The Good and the Bad of Angular Development [Електронний ресурс]. – Режим доступу: https://goo.gl/GuZSXM.

29. Диаграмма компонентов (component diagram) [Електронний ресурс]. – Режим доступу до ресурсу: https://goo.gl/WB4JMy.

30. Аналіз якості програмного забезпечення [Електронний ресурс]. – Режим доступу до ресурсу: https://goo.gl/hNrLCO.

31. Характеристики якості ПЗ [Електронний ресурс]. – Режим доступу до ресурсу: https://goo.gl/xKHyXB.

32. Тестування програмного забезпечення [Електронний ресурс]. Режим доступу до ресурсу: https://goo.gl/K87uTM.

33. Функциональное тестирование или Functional Testing [Електронний ресурс]. – Режим доступу до ресурсу: https://goo.gl/gnesYs.

34. Шаблон ціннісної пропозиції [Електронний ресурс]. – Режим доступу: https://goo.gl/k6pQga.

35. Вимоги до програмного забезпечення [Електронний ресурс]. – Режим доступу: https://goo.gl/3b772H.

36. Нефункціональні вимоги [Електронний ресурс]. – Режим доступу: https://goo.gl/TTAJpN.

37. C Sharp [Електронний ресурс]. – Режим доступу: https://goo.gl/VAuv3p.

38. SQL Injection Attacks Are Rampant: How to Stop Your Next Hack Attack [Електронний ресурс]. – Режим доступу: https://goo.gl/RxnbWp.

39. SQL-инъекция [Електронний ресурс]. – Режим доступу: https://goo.gl/UEB2FN.

40. КАК ПОСТРОИТЬ «ДЕРЕВО ПРОБЛЕМ»? [Електронний ресурс]. – Режим доступу: https://goo.gl/97E8Ln.

41. Глава 2 Стандарт ANSI PMI PMBOK. Заинтересованные стороны и основные действующие лица проекта [Електронний ресурс]. Режим доступу: https://goo.gl/7d4ixh.

42. Работа с заинтересованными сторонами проекта [Електронний ресурс]. – Режим доступу: https://goo.gl/gtqafn.

43. Анализ заинтересованных сторон проекта [Електронний ресурс]. – Режим доступу: https://goo.gl/7itrdh.

44. Понятие заинтересованных лиц в проекте [Електронний ресурс]. – Режим доступу: https://goo.gl/qj3npq.

45. Сегментация рынка [Електронний ресурс]. – Режим доступу: https://goo.gl/m6e2Mi.

46. Сегментация рынка [Електронний ресурс]. – Режим доступу: https://goo.gl/Qn4EHZ.

47. Выручка, доход и прибыль [Електронний ресурс]. – Режим доступу: https://goo.gl/1eSx9a.

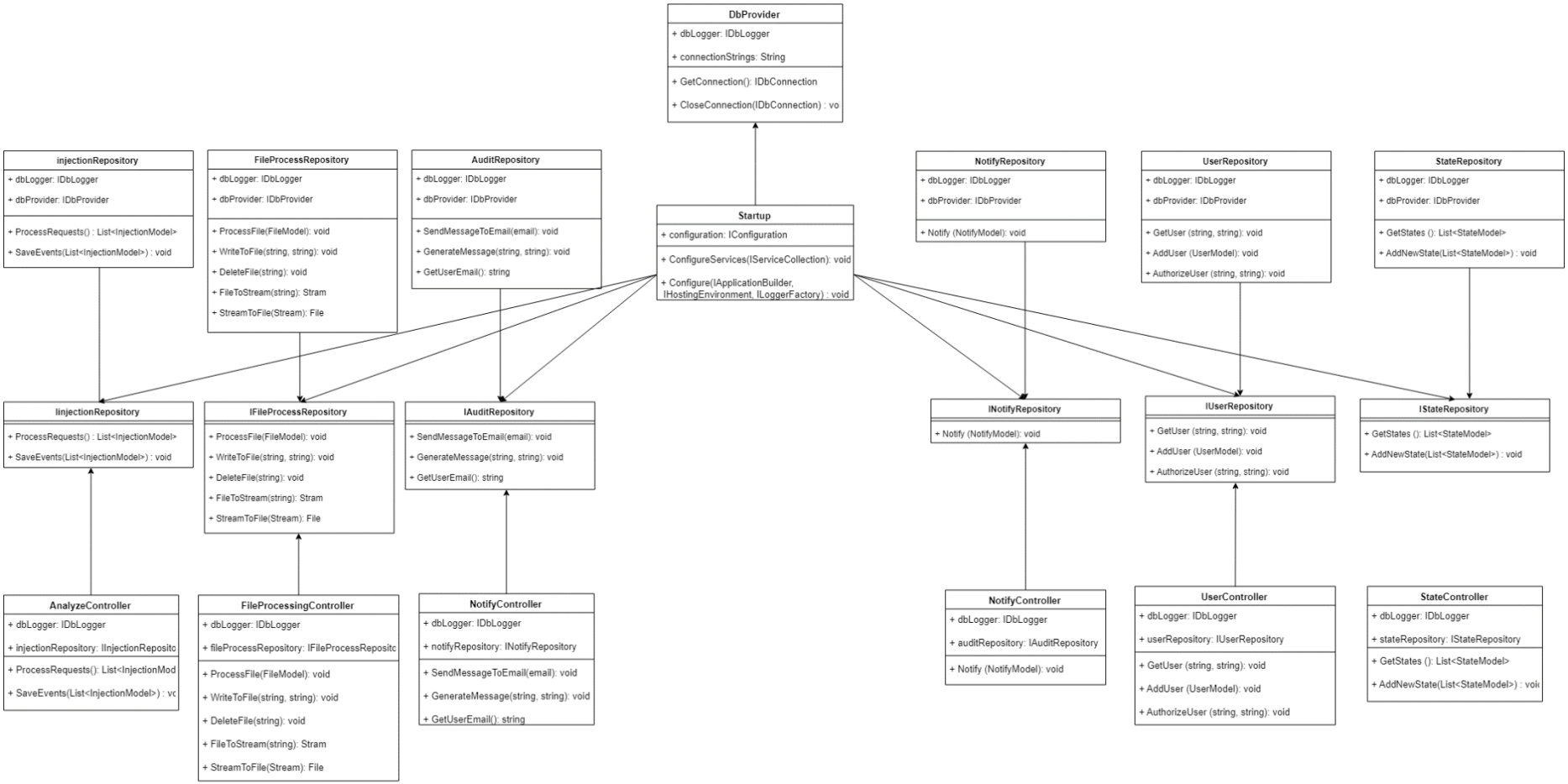
48. Доход и прибыль [Електронний ресурс]. – Режим доступу: https://goo.gl/rSuXEX.

49. Мета і стратегія розробки проекту та його життєвий цикл – [Електронний ресурс]. – Режим доступу: https://goo.gl/upuofB.

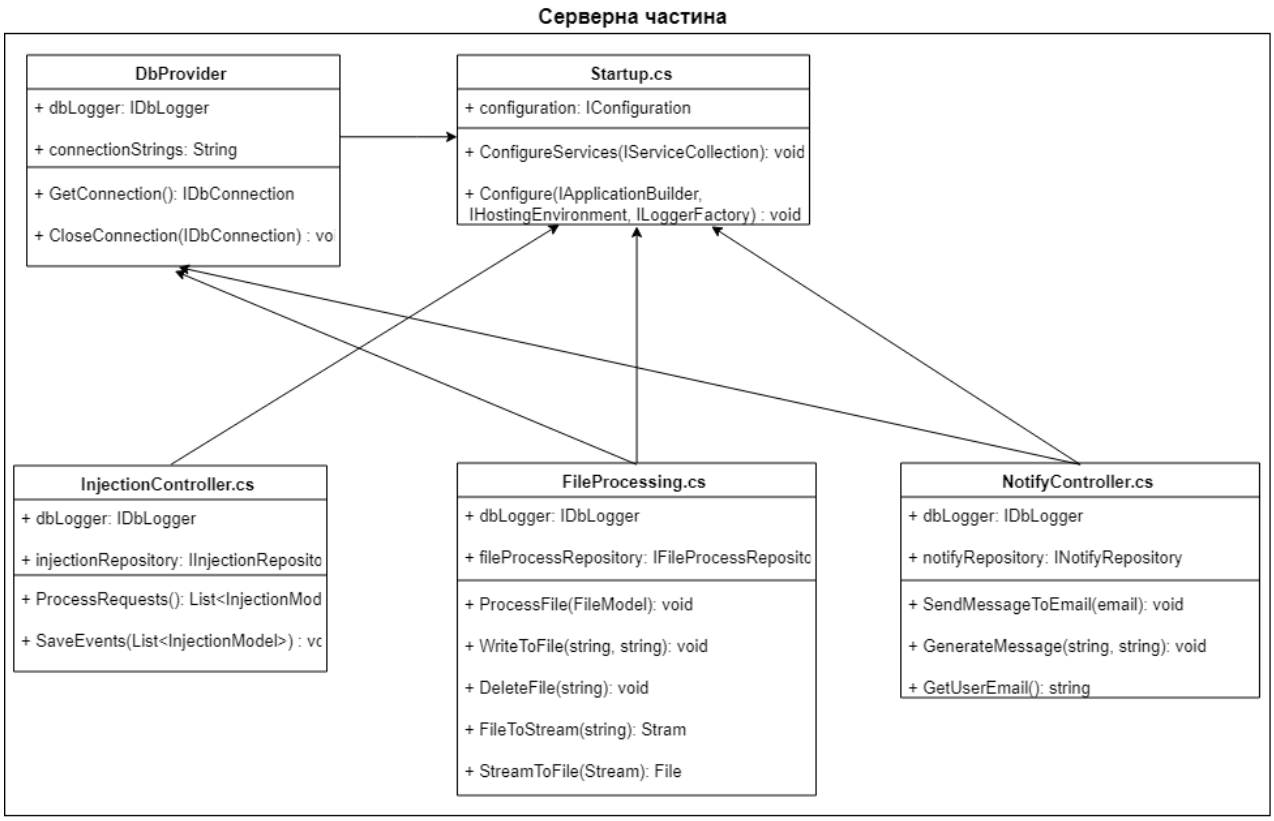
50. Створення ціннісної пропозиції [Електронний ресурс]. – Режим доступу: https://goo.gl/6E8aYG.

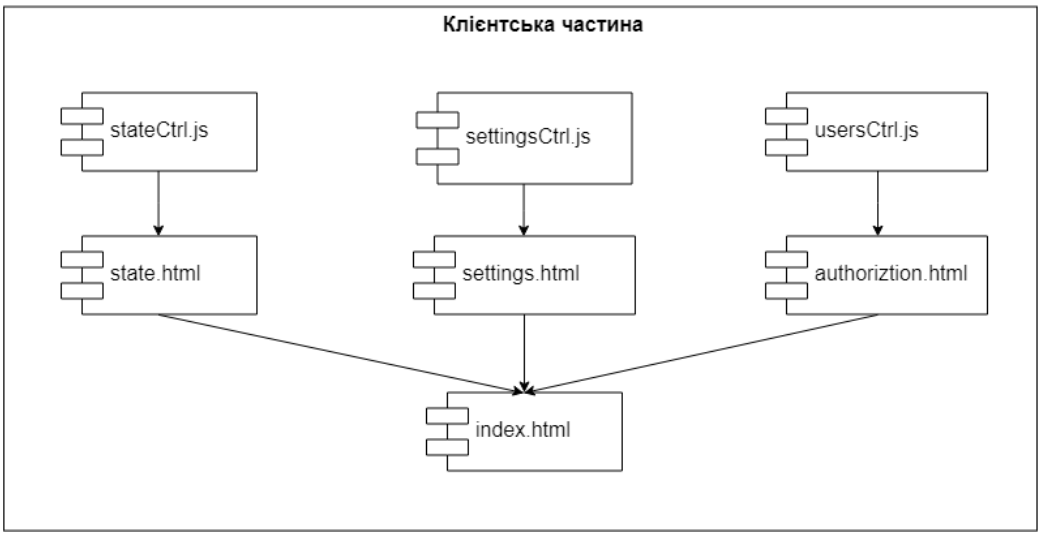
ДОДАТОК 1

Графічні матеріали



Діаграма класів
Васін Є.В., КП-71мп





Діаграма компонентів
Васін Є.В., КП-71мп

ДОДАТОК 2

Фрагмент тексту програми

ClientsController.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace InjectionTool.API.Controllers
{
    [Route("api/clients")]
    public class ClientsController : ControllerBase
    {
        private readonly IClientsRepository _clientsRepository;
        public                    ClientsController(IClientsRepository
clientsRepository)
        {
            _clientsRepository = clientsRepository;
        }

        [HttpGet]
        [Route("")]
        public async Task<List<Client>> GetTemplatesAsync(string id)
        {
            return await _clientsRepository.GetClientsAsync(id);
        }
    }
}
```

MenusController.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
```

```
using System.Text;
using System.Threading.Tasks;

namespace InjectionTool.API.Controllers
{
    [Route("api/menus")]
    public class MenusController : ControllerBase
    {
        private readonly IMenusRepository _menusRepository;
        public MenusController(IMenusRepository menusRepository)
        {
            _menusRepository = menusRepository;
        }

        [HttpGet]
        [Route("")]
        public async Task<List<Menu>> GetMenusAsync()
        {
            return await _menusRepository.GetMenusAsync();
        }


        [HttpPut]
        [Route("")]
        public                async                Task<List<Menu>>
PutRequestAsync([FromBody]List<Menu> menus)
        {
            return await _menusRepository.PutMenusAsync(menus);
        }
    }
}
```

NewsController.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```
namespace InjectionTool.API.Controllers
{
    [Route("api/News")]
    public class NewsController : ControllerBase
    {
        private readonly INewsRepository _newsRepository;
        public NewsController(INewsRepository newsRepository)
        {
            _newsRepository = newsRepository;
        }

        [HttpPost]
        [Route("")]
        public async Task PostNewsAsync([FromBody]New news)
        {
            await _newsRepository.PostNewsAsync(news);
        }
    }
}
```

NotificationsController.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace InjectionTool.API.Controllers
{
    [Route("api/notifications")]
    public class NotificationsController : ControllerBase
    {
        private readonly INotificationsRepository
_notificationsRepository;
        public NotificationsController(INotificationsRepository
notificationsRepository)
        {
            _notificationsRepository = notificationsRepository;
        }
```

```
        [HttpGet]
        [Route("")]
        public             async             Task<List<Notification>>
    GetNewNotificationsAsync()
        {
            return                                                  await
    _notificationsRepository.GetNewNotificationsAsync();
        }

        [HttpPost]
        [Route("")]
        public                 async                Task<Notification>
    PostRequestAsync([FromBody]Notification notification)
        {
            return                                                  await
    _notificationsRepository.PostNotificationAsync(notification);
        }

        [HttpPut]
        [Route("{notificationId}")]
        public    async    Task<Notification>    PutRequestAsync(int
    notificationId, [FromBody]Notification notification)
        {
            return                                                  await
    _notificationsRepository.PutNotificationAsync(notificationId,
    notification);
        }
    }
}
```

RequestsController.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace InjectionTool.API.Controllers
```

```
{
    [Route("api/requests")]
    public class RequestsController : ControllerBase
    {
        private readonly IRequestsRepository _requestsRepository;
        public                  RequestsController(IRequestsRepository
requestsRepository)
        {
            _requestsRepository = requestsRepository;
        }

        [HttpGet]
        [Route("")]
        public async Task<List<Request>> GetAllRequestsAsync()
        {
            var             requests             =             await
_requestsRepository.GetAllRequestsAsync();
            return requests;
        }

        [HttpGet]
        [Route("{requestId}")]
        public async Task<Request> GetRequestAsync(int requestId)
        {
            return                                                  await
_requestsRepository.GetRequestAsync(requestId);
        }

        [HttpPost]
        [Route("")]
        public async Task<Request> PostRequestAsync([FromBody]Request
request)
        {
            return                                                  await
_requestsRepository.PostRequestAsync(request);
        }

        [HttpDelete]
        [Route("{requestId}")]
        public async Task DeleteRequestAsync(int requestId)
        {
            await _requestsRepository.DeleteRequestAsync(requestId);
```

```
        }

        [HttpPut]
        [Route("{requestId}")]
        public async Task<Request> PutRequestAsync(int requestId,
[FromBody]Request request)
        {
            return                                                  await
_requestsRepository.PutRequestAsync(requestId, request);
        }

        [HttpPut]
        [Route("{requestId}/markasdanger")]
        public   async   Task<Request>   MarkRequestAsDangerAsync(int
requestId)
        {
            return                                                  await
_requestsRepository.MarkRequestAsDangerAsync(requestId);
        }
    }
}
```

SettingsController.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace InjectionTool.API.Controllers
{
    [Route("api/settings")]
    public class SettingsController : ControllerBase
    {
        private readonly ISettingsRepository _settingsRepository;
        public                  SettingsController(ISettingsRepository
settingsRepository)
        {
            _settingsRepository = settingsRepository;
```

```
        }

        [HttpGet]
        [Route("")]
        public async Task<Settings> GetSettingsAsync()
        {
            return await _settingsRepository.GetSettingsAsync();
        }


        [HttpPut]
        [Route("")]
        public                  async                  Task<Settings>
PutRequestAsync([FromBody]Settings settings)
        {
            return                                                  await
_settingsRepository.PutSettingsAsync(settings);
        }
    }

}
```

TemplatesController.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace InjectionTool.API.Controllers
{

    [Route("api/templates")]
    public class TemplatesController : ControllerBase
    {
        private readonly ITemplatesRepository _templatesRepository;
        public               TemplatesController(ITemplatesRepository
templatesRepository)
        {
```

```
            _templatesRepository = templatesRepository;
        }

        [HttpGet]
        [Route("")]
        public async Task<List<Template>> GetTemplatesAsync()
        {
            return await _templatesRepository.GetTemplatesAsync();
        }


        [HttpDelete]
        [Route("{templateId}")]
        public async Task DeleteTemplateAsync(int templateId)
        {
            await
_templatesRepository.DeleteTemplateAsync(templateId);
        }

        [HttpGet]
        [Route("create/{requestId}")]
        public async Task CreateRequestTemplate(int requestId)
        {
            await
_templatesRepository.CreateRequestTemplate(requestId);
        }
    }
}
```

UsersController.cs

```
using System;
using Microsoft.AspNetCore.Mvc;
using Bone.Domain.Repositories;
using System.Threading.Tasks;
using Bone.Domain.Entities;

namespace InjectionTool.API.Controllers
{
    [Route("api/users")]
    public class UsersController : ControllerBase
    {
```

```
            private readonly IUsersRepository _usersRepository;
            public UsersController(IUsersRepository usersRepository)
            {
                _usersRepository = usersRepository;
            }

            [HttpGet]
            [Route("")]
            public async Task<UserProfile> GetUserProfileAsync()
            {
                return await _usersRepository.GetUserProfileAsync();
            }

            [HttpPut]
            [Route("{id}")]
            public async Task<UserProfile> PutUserProfileAsync(int id,
    [FromBody]UserProfile userProfile)
            {
                return await _usersRepository.PutUserProfileAsync(id,
    userProfile);
            }
        }
}
```

Startup.cs

```
    using System;
    using System.IO;
    using Bone.Data.Repositories;
    using Bone.Domain.Repositories;
    using Microsoft.AspNetCore.Builder;
    using Microsoft.AspNetCore.Hosting;
    using Microsoft.Extensions.Configuration;
    using Microsoft.Extensions.DependencyInjection;
    using Microsoft.Extensions.Logging;
    using Swashbuckle.AspNetCore.Swagger;
    using Newtonsoft.Json.Converters;
    using Bone.Data.Middleware.Providers.Logger;
    using System.Net;
    using System.Security.Cryptography.X509Certificates;
    using System.Net.Security;
    using Newtonsoft.Json.Serialization;
```

```
namespace InjectionTool.API
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        private void SslValidation()
        {
            ServicePointManager.ServerCertificateValidationCallback =
delegate (object s, X509Certificate certificate, X509Chain chain,
SslPolicyErrors sslPolicyErrors) { return true; };
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to
add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {

            string                  connectionString                  =
Configuration.GetConnectionString("DefaultConnection");
            ILoggerFactory loggerFactory = new LoggerFactory();
            LogLevel                fileCustomLogLevel                =
Configuration.GetSection("Logging:FileLogger:LogLevel").GetValue<LogL
evel>("Default");
            LogLevel                    DBLogLevel                    =
Configuration.GetSection("Logging:DbLogger:LogLevel").GetValue<LogLev
el>("Default");
            if  (!Directory.Exists(Directory.GetCurrentDirectory()  +
"\\logs\\"))

Directory.CreateDirectory(Directory.GetCurrentDirectory()           +
"\\logs\\");


loggerFactory.AddFile(Path.Combine(Directory.GetCurrentDirectory()  +
"\\logs\\",
```

```
                                    DateTime.Now.Date.ToString("dd-MM-
yyyy") + "-custom.txt"),
                                    fileCustomLogLevel);

            loggerFactory.AddDB(connectionString, DBLogLevel);

            services.AddTransient<IUsersRepository,
UsersRepository>(provider  =>  new  UsersRepository(connectionString,
loggerFactory, Configuration));
            services.AddTransient<IRequestsRepository,
RequestsRepository>(provider                    =>                    new
RequestsRepository(connectionString, loggerFactory, Configuration));
            services.AddTransient<INotificationsRepository,
NotificationsRepository>(provider                 =>                 new
NotificationsRepository(connectionString,               loggerFactory,
Configuration));
            services.AddTransient<ISettingsRepository,
SettingsRepository>(provider                    =>                    new
SettingsRepository(connectionString, loggerFactory, Configuration));
            services.AddTransient<IMenusRepository,
MenusRepository>(provider  =>  new  MenusRepository(connectionString,
loggerFactory, Configuration));
            services.AddTransient<ITemplatesRepository,
TemplatesRepository>(provider                   =>                    new
TemplatesRepository(connectionString, loggerFactory, Configuration,
                new               RequestsRepository(connectionString,
loggerFactory, Configuration)));
            services.AddTransient<IClientsRepository,
ClientsRepository>(provider                    =>                    new
ClientsRepository(connectionString, loggerFactory, Configuration));
            services.AddTransient<INewsRepository,
NewsRepository>(provider   =>   new   NewsRepository(connectionString,
loggerFactory, Configuration,
                new          NotificationsRepository(connectionString,
loggerFactory, Configuration),
                new               RequestsRepository(connectionString,
loggerFactory, Configuration)));

            services.AddMvc().AddJsonOptions(options =>
                options.SerializerSettings.Converters.Add(new
StringEnumConverter())
            );
```

```
            services.AddMvc().AddJsonOptions(options =>
                options.SerializerSettings.ContractResolver = new
DefaultContractResolver()
            );

            services.AddCors(options =>
            {
                options.AddPolicy("AllowAll",
                    builder =>
                    {
                        builder
                        .AllowAnyOrigin()
                        .AllowAnyMethod()
                        .AllowAnyHeader()
                        .AllowCredentials();
                    });
            });

            // Register the Swagger generator, defining 1 or more
Swagger documents
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new Info { Title = "Bone API",
Version = "v1" });
            });
        }

        // This method gets called by the runtime. Use this method to
configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app,
IHostingEnvironment env, ILoggerFactory loggerFactory)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseCors("AllowAll");

            loggerFactory.CreateLogger("FileLogger");
```

```
                app.UseSwagger();
                app.UseSwaggerUI(c =>
                {
                    c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API
V1");
                });

                app.UseMvc();
            }
        }
}
```

BaseRepository.cs

```
using Bone.Data.Middleware;
using Microsoft.Extensions.Logging;
using MySql.Data.MySqlClient;
using System;
using System.Data;
using System.IO;

namespace Bone.Data.Repositories
{
    public abstract class BaseRepository
    {
        protected string connectionString = null;
        protected ILogger fileLogger;
        public     BaseRepository(string     conn,     ILoggerFactory
loggerFactory)
        {
            connectionString = conn;
            fileLogger                                                =
loggerFactory.CreateLogger("RepositoryLogger");
        }
        public IDbConnection GetConnection()
        {
            var conn = new MySqlConnection(connectionString);
            return conn;
        }
    }
}
```

ClientsRepository.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

namespace Bone.Data.Repositories
{

    public class ClientsRepository : BaseRepository, IClientsRepository
    {
        public IConfiguration Configuration { get; }
        public ClientsRepository(string connection, ILoggerFactory loggerFactory, IConfiguration configuration) : base(connection, loggerFactory)
        {
            Configuration = configuration;
        }

        public async Task<List<Client>> GetClientsAsync(string id)
        {
            return null;
        }

        public void Dispose()
        {

        }
    }
}
```

MenusRepository.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Dapper;
using Microsoft.Extensions.Configuration;
```

```
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Threading.Tasks;

namespace Bone.Data.Repositories
{
    public class MenusRepository : BaseRepository, IMenusRepository
    {
        public IConfiguration Configuration { get; }
        public MenusRepository(string connection, ILoggerFactory
loggerFactory, IConfiguration configuration) : base(connection,
loggerFactory)
        {
            Configuration = configuration;
        }

        public async Task<List<Menu>> GetMenusAsync()
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                List<Menu> menus = await GetMenusAsync(connection);
                return menus;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public async Task<List<Menu>> PutMenusAsync(List<Menu> menus)
        {
            var connection = GetConnection();
            try
```

```
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                await UpdateMenusAsync(menus, connection);

                List<Menu> _menus = await GetMenusAsync(connection);
                return menus;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        private async Task<List<Menu>> GetMenusAsync(IDbConnection connection)
        {
            string sqlQuery = @"SELECT id as Id, menu_name as MenuName, show_menu as ShowMenu, menu_title as MenuTitle FROM menus";

            List<Menu> menus = (await connection.QueryAsync<Menu>(sqlQuery)).ToList();
            return menus;
        }

        private async Task UpdateMenusAsync(List<Menu> menus, IDbConnection connection)
        {
            string sqlQuery = @"update menus set show_menu = @p_show_menu where id = @p_id";

            foreach (Menu menu in menus)
            {
                DynamicParameters parameters = new DynamicParameters();
                parameters.Add("p_show_menu", menu.ShowMenu, DbType.Boolean, direction: ParameterDirection.Input);
                parameters.Add("p_id", menu.Id, DbType.Int16, direction: ParameterDirection.Input);
```

```
                    await connection.ExecuteAsync(sqlQuery, parameters);
                }
            }


            public void Dispose()
            {

            }
        }
}
```

NotificationsRepository.cs

```
using System;
using Dapper;
using System.Data;
using System.Linq;
using Microsoft.Extensions.Logging;
using Bone.Data.Middleware;
using Bone.Domain.Repositories;
using Microsoft.Extensions.Configuration;
using Bone.Domain.Entities;
using System.Threading.Tasks;
using System.Collections.Generic;

namespace Bone.Data.Repositories
{
    public class NotificationsRepository : BaseRepository,
INotificationsRepository
    {
        public IConfiguration Configuration { get; }
        public NotificationsRepository(string connection,
ILoggerFactory loggerFactory, IConfiguration configuration) :
base(connection, loggerFactory)
        {
            Configuration = configuration;
        }

        public async Task<List<Notification>>
GetNewNotificationsAsync()
```

```
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }
                string sqlQuery = @"SELECT n.id as Id,
                                           n.message as Message,
                                           n.is_read as IsRead
                                    FROM notifications n
                                    where n.is_read = 0";

                List<Notification>      userProfile      =      (await
connection.QueryAsync<Notification>(sqlQuery)).ToList();
                return userProfile;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public                async                Task<Notification>
PostNotificationAsync(Notification notification)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                int          notificationId          =          await
AddRequestAsync(notification, connection);
                Notification       newNotification       =       await
GetNotificationAsync(notificationId, connection);

                return newNotification;
            }
```

```
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public async Task<Notification> PutNotificationAsync(int
notificationId, Notification notification)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                Notification _notification = await
GetNotificationAsync(notificationId, connection);
                _notification = await
UpdateNotificationAsync((int)_notification.Id, notification,
connection);

                return _notification;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        private async Task<Notification> GetNotificationAsync(int
notificationId, IDbConnection connection)
        {
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_notificationId", notificationId,
DbType.Int32, direction: ParameterDirection.Input);

            string sqlQuery = @"SELECT n.id as Id,
                                       n.message as Message,
                                       n.is_read as IsRead
                                FROM notifications n
```

```
                                where n.id = @p_notificationId";

            Notification          request          =          (await
connection.QueryAsync<Notification>(sqlQuery,
parameters)).FirstOrDefault()
                ?? throw new Exception("Request not found");

            return request;
        }

        private    async    Task<int>    AddRequestAsync(Notification
notification, IDbConnection connection)
        {
            string sqlQuery = @"select IFNULL(max(n.id), 0)+1
                                from notifications n";
            int                          newId                          =
connection.Query<int>(sqlQuery).FirstOrDefault();

            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_id",  newId,  DbType.Int32,  direction:
ParameterDirection.Input);
            parameters.Add("p_message",          notification.Message,
DbType.String, direction: ParameterDirection.Input);
            parameters.Add("p_is_read",          notification.IsRead,
DbType.Int16, direction: ParameterDirection.Input);

            sqlQuery  =  @"INSERT  INTO  notifications  (id,  message,
is_read) VALUES (@p_id, @p_message, @p_is_read)";
            await connection.ExecuteAsync(sqlQuery, parameters);

            return newId;
        }

        private async Task<Notification> UpdateNotificationAsync(int
notificationId, Notification notification, IDbConnection connection)
        {
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_notification_id",        notificationId,
DbType.Int32, direction: ParameterDirection.Input);
            parameters.Add("p_message",          notification.Message,
DbType.String, direction: ParameterDirection.Input);
```

```
            parameters.Add("p_is_read", notification.IsRead,
DbType.Int16, direction: ParameterDirection.Input);

            string sqlQuery = @"UPDATE notifications n SET n.message
=  @p_message,  n.is_read  =  @p_is_read  WHERE  n.id  =
@p_notification_id";

            await connection.ExecuteAsync(sqlQuery, parameters);

            Notification _request = await
GetNotificationAsync(notificationId, connection);
            return _request;
        }

        public void Dispose()
        {

        }


    }
}
```

RequestsRepository.cs

```
using Bone.Data.Middleware;
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Dapper;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Bone.Data.Repositories
{
    public class RequestsRepository : BaseRepository,
IRequestsRepository
```

```
    {
        public IConfiguration Configuration { get; }
        public RequestsRepository(string connection, ILoggerFactory
loggerFactory, IConfiguration configuration) : base(connection,
loggerFactory)
        {
            Configuration = configuration;
        }

        public async Task<List<Request>> GetAllRequestsAsync()
        {
            DateTime date = DateTime.Now;
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                List<Request>       requestsList       =       await
GetAllRequestsAsync(connection);
                return requestsList;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public async Task<Request> GetRequestAsync(int requestId)
        {
            DateTime date = DateTime.Now;
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }
```

```
                Request request = await GetRequestAsync(requestId,
connection);
                return request;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public async Task<Request> PostRequestAsync(Request request)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                int requestId = await AddRequestAsync(request,
connection);
                Request newRequest = await GetRequestAsync(requestId,
connection);

                return newRequest;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }


        public async Task DeleteRequestAsync(int requestId)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
```

```
                }

                Request request = await GetRequestAsync(requestId,
connection);
                if (request.Parameters.Count > 0 &&
request.Parameters != null)
                {
                    await
DeleteRequestParametersAsync((int)request.Id, connection);
                }
                await DeleteRequestAsync((int)request.Id,
connection);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public async Task<Request> PutRequestAsync(int requestId,
Request request)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                Request _request = await GetRequestAsync(requestId,
connection);
                _request = await UpdateRequestAsync((int)_request.Id,
request, connection);

                return _request;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
```

```
        public   async   Task<Request>   MarkRequestAsDangerAsync(int
requestId)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                Request  _request  =  await  GetRequestAsync(requestId,
connection);
                await      MarkRequestAsDangerAsync((int)_request.Id,
connection);

                return _request;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        private  async  Task  MarkRequestAsDangerAsync(int  requestId,
IDbConnection connection)
        {
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_request_id",  requestId,  DbType.Int32,
direction: ParameterDirection.Input);

            string  sqlQuery  =  @"UPDATE  requests  r  SET  r.type  =
'DANGER' WHERE r.id = @p_request_id";

            await connection.ExecuteAsync(sqlQuery, parameters);
        }

        private                async                Task<List<Request>>
GetAllRequestsAsync(IDbConnection connection)
        {
            string sqlQuery = @"select r.id as Id,
```

```
                                        r.url as Url,
                                        r.method as Method,
                                    r.type as RequestType,
r.reason as Reason
                                from requests r
order by r.id desc";
            List<Request>               requestsList               =
connection.Query<Request>(sqlQuery).ToList();

            foreach (Request request in requestsList)
            {
                request.Parameters              =              await
GetRequestParametersAsync((int)request.Id, connection);
            }

            return requestsList;
        }

        private   async   Task   DeleteRequestAsync(int   requestId,
IDbConnection connection)
        {
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_requestId",   requestId,   DbType.Int32,
direction: ParameterDirection.Input);

            string sqlQuery = @"delete from requests
                                where id = @p_requestId";

            await connection.ExecuteAsync(sqlQuery, parameters);
        }

        private    async    Task    DeleteRequestParametersAsync(int
requestId, IDbConnection connection)
        {
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_requestId",   requestId,   DbType.Int32,
direction: ParameterDirection.Input);

            string sqlQuery = @"delete from requests_parameters
                                where request_id = @p_requestId";

            await connection.ExecuteAsync(sqlQuery, parameters);
```

```
        }

        private async Task<int> AddRequestAsync(Request request,
IDbConnection connection)
        {
            string sqlQuery = @"select IFNULL(max(r.id), 0)+1
                                from requests r";
            int                         newId                         =
connection.Query<int>(sqlQuery).FirstOrDefault();

            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_id",  newId,  DbType.Int32,  direction:
ParameterDirection.Input);
            parameters.Add("p_url",    request.Url,    DbType.String,
direction: ParameterDirection.Input);
            parameters.Add("p_method", request.Method, DbType.String,
direction: ParameterDirection.Input);
            parameters.Add("p_type",             request.RequestType,
DbType.String, direction: ParameterDirection.Input);
            parameters.Add("p_reason", request.Reason, DbType.String,
direction: ParameterDirection.Input);

            sqlQuery = @"INSERT INTO requests (id, url, method, type,
reason) VALUES (@p_id, @p_url, @p_method, @p_type, @p_reason)";
            await connection.ExecuteAsync(sqlQuery, parameters);

            if       (request.Parameters       !=       null       &&
request.Parameters.Count > 0)
            {
                await                AddRequestParametersAsync(newId,
request.Parameters, connection);
            }

            return newId;
        }

        private async Task AddRequestParametersAsync(int requstId,
List<RequestParameter> requestParameters, IDbConnection connection)
        {
            string sqlQuery = @"select IFNULL(max(r.id), 0)+1
                                from requests_parameters r";
```

```
            int                         newId                       =
connection.Query<int>(sqlQuery).FirstOrDefault();

            foreach     (RequestParameter     requestParameter     in
requestParameters)
            {
                DynamicParameters       parameters       =       new
DynamicParameters();
                parameters.Add("p_id",      newId,      DbType.Int32,
direction: ParameterDirection.Input);
                parameters.Add("p_key",
requestParameter.ParameterKey,        DbType.String,        direction:
ParameterDirection.Input);
                parameters.Add("p_value",
requestParameter.ParameterValue,       DbType.String,       direction:
ParameterDirection.Input);
                parameters.Add("p_requestId", requstId, DbType.Int32,
direction: ParameterDirection.Input);

                sqlQuery = @"INSERT INTO requests_parameters (`id`,
`key`,  `value`,  `request_id`)  VALUES  (@p_id,  @p_key,  @p_value,
@p_requestId)";
                await connection.ExecuteAsync(sqlQuery, parameters);

                newId += 1;
            }
        }

        private  async  Task<Request>  GetRequestAsync(int  requestId,
IDbConnection connection)
        {
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_requestId",   requestId,   DbType.Int32,
direction: ParameterDirection.Input);

            string sqlQuery = @"select r.id as Id,
                                       r.url as Url,
                                       r.method as Method,
                                       r.type as RequestType,
                                       r.reason as Reason
                                  from requests r
                                 where r.id = @p_requestId";
```

```
            Request request = connection.Query<Request>(sqlQuery,
parameters).FirstOrDefault();

            if (request != null)
            {
                request.Parameters = await
GetRequestParametersAsync(requestId, connection);
            }
            else
            {
                throw new Exception("Request not found");
            }

            return request;
        }

        private async Task<List<RequestParameter>>
GetRequestParametersAsync(int requestId, IDbConnection connection)
        {
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_requestId", requestId, DbType.Int32,
direction: ParameterDirection.Input);

            string sqlQuery = @"select r.key as ParameterKey,
                                       r.value as ParameterValue
                                from requests_parameters r
                                where r.request_id = @p_requestId";

            List<RequestParameter> requestParameters = (await
connection.QueryAsync<RequestParameter>(sqlQuery,
parameters)).ToList();
            return requestParameters;
        }

        private async Task<Request> UpdateRequestAsync(int requestId,
Request request, IDbConnection connection)
        {
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_request_id", request.Id, DbType.Int32,
direction: ParameterDirection.Input);
            parameters.Add("p_url", request.Url, DbType.String,
direction: ParameterDirection.Input);
```

```
            parameters.Add("p_method", request.Method, DbType.String,
        direction: ParameterDirection.Input);
            parameters.Add("p_type",             request.RequestType,
        DbType.String, direction: ParameterDirection.Input);

            string sqlQuery = @"UPDATE requests r SET r.method =
        @p_method, r.url = @p_url, r.type = @p_type WHERE r.id =
        @p_request_id";

            await connection.ExecuteAsync(sqlQuery, parameters);

            await             UpdateRequestParametersAsync(requestId,
        request.Parameters, connection);

            Request _request = await GetRequestAsync(requestId,
        connection);
            return _request;
        }

        private async Task UpdateRequestParametersAsync(int
        requestId, List<RequestParameter> requestParameters, IDbConnection
        connection)
        {
            await             DeleteRequestParametersAsync(requestId,
        connection);
            await                AddRequestParametersAsync(requestId,
        requestParameters, connection);
        }

        public void Dispose()
        {

        }
    }
}
```

SettingsRepository.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Dapper;
using Microsoft.Extensions.Configuration;
```

```
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Bone.Data.Repositories
{
    public class SettingsRepository : BaseRepository, ISettingsRepository
    {
        public IConfiguration Configuration { get; }
        public SettingsRepository(string connection, ILoggerFactory loggerFactory, IConfiguration configuration) : base(connection, loggerFactory)
        {
            Configuration = configuration;
        }

        public async Task<Settings> GetSettingsAsync()
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                Settings settings = await GetSettingsAsync(connection);
                return settings;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
```

```
        public    async    Task<Settings>    PutSettingsAsync(Settings
settings)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                List<Setting>              settingsList              =
MapSettingsToList(settings);
                await UpdateSettingsAsync(settingsList, connection);

                Settings          _settings          =          await
GetSettingsAsync(connection);
                return _settings;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        private List<Setting> MapSettingsToList(Settings settings)
        {
            List<Setting> settingsList = new List<Setting>()
            {
                new Setting()
                {
                    ParameterName = "GetNotifications",
                    ParameterValue   =   settings.GetNotifications   ?
"true" : "false"
                },
                new Setting()
                {
                    ParameterName = "MakeBackUp",
                    ParameterValue  =  settings.MakeBackUp  ?  "true"  :
"false"
                },
                new Setting()
```

```
                {
                    ParameterName = "NotifyEmail",
                    ParameterValue = settings.NotifyEmail
                },
                new Setting()
                {
                    ParameterName = "PeriodBackUp",
                    ParameterValue = settings.PeriodBackUp.ToString()
                },
                new Setting()
                {
                    ParameterName = "TimeToBackUp",
                    ParameterValue = settings.TimeToBackUp
                }
            };
            return settingsList;
        }

        private async Task<Settings> GetSettingsAsync(IDbConnection
connection)
        {
            string sqlQuery = @"SELECT parameter_name as
ParameterName, parameter_value as ParameterValue FROM settings";

            List<Setting> settingsDict = (await
connection.QueryAsync<Setting>(sqlQuery)).ToList();
            Settings settings = new Settings();

            settings.GetNotifications =
Convert.ToBoolean(settingsDict.FirstOrDefault(setting =>
setting.ParameterName == "GetNotifications")?.ParameterValue);
            settings.MakeBackUp =
Convert.ToBoolean(settingsDict.FirstOrDefault(setting =>
setting.ParameterName == "MakeBackUp")?.ParameterValue);
            settings.NotifyEmail =
settingsDict.FirstOrDefault(setting => setting.ParameterName ==
"NotifyEmail")?.ParameterValue;
            settings.PeriodBackUp =
Convert.ToInt16(settingsDict.FirstOrDefault(setting =>
setting.ParameterName == "PeriodBackUp")?.ParameterValue);
```

```
            settings.TimeToBackUp                                    =
    settingsDict.FirstOrDefault(setting   =>   setting.ParameterName   ==
    "TimeToBackUp")?.ParameterValue;

            return settings;
        }

        private   async   Task   UpdateSettingsAsync(List<Setting>
    settings, IDbConnection connection)
        {
            string sqlQuery = @"update settings set parameter_value =
    @p_parameter_value where parameter_name = @p_parameter_name";

            foreach(Setting setting in settings)
            {
                DynamicParameters       parameters       =       new
    DynamicParameters();
                parameters.Add("p_parameter_value",
    setting.ParameterValue,           DbType.String,           direction:
    ParameterDirection.Input);
                parameters.Add("p_parameter_name",
    setting.ParameterName,            DbType.String,           direction:
    ParameterDirection.Input);

                await connection.ExecuteAsync(sqlQuery, parameters);
            }
        }


        public void Dispose()
        {

        }
    }
}
```

TemplatesRepository.cs

```
using Bone.Domain.Entities;
using Bone.Domain.Repositories;
using Dapper;
using Microsoft.Extensions.Configuration;
```

```
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Bone.Data.Repositories
{
    public class TemplatesRepository : BaseRepository, ITemplatesRepository
    {
        public IConfiguration Configuration { get; }

        private readonly IRequestsRepository _requestsRepository;
        public TemplatesRepository(string connection, ILoggerFactory loggerFactory, IConfiguration configuration, IRequestsRepository requestsRepository) : base(connection, loggerFactory)
        {
            Configuration = configuration;
            _requestsRepository = requestsRepository;
        }

        public async Task<List<Template>> GetTemplatesAsync()
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                List<Template> templates = await GetTemplatesAsync(connection);
                return templates;
            }
            catch (Exception ex)
            {
                throw ex;
            }
```

```
        }

        public async Task DeleteTemplateAsync(int templateId)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                await DeleteTemplateAsync(templateId, connection);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public async Task CreateRequestTemplate(int requestId)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                await CreateRequestTemplate(requestId, connection);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        private async Task CreateRequestTemplate(int reqeustId,
IDbConnection connection)
        {
```

```
            Request request = await
_requestsRepository.GetRequestAsync(reqeustId);
            string[] words =
request.Parameters.FirstOrDefault().ParameterValue.Split(" ");
            string template = string.Empty;
            foreach (string word in words)
            {
                if (word.ToUpper() != "SELECT"
                    && word.ToUpper() != "DELETE"
                    && word.ToUpper() != "INSERT"
                    && word.ToUpper() != "UPDATE"
                    && word.ToUpper() != "OR"
                    && word.ToUpper() != "AND"
                    && word.ToUpper() != "="
                    && word.ToUpper() != "!=")
                {
                    template += @" ([^\s]+)";
                }
                else
                {
                    template += $" {word}";
                }
            }
            DynamicParameters parameters = new DynamicParameters();
            parameters.Add("p_url",
"http://localhost:54082/api/news",      DbType.String,      direction:
ParameterDirection.Input);
            parameters.Add("p_param_template",              template,
DbType.String, direction: ParameterDirection.Input);
            parameters.Add("p_method",      "POST",      DbType.String,
direction: ParameterDirection.Input);

            string sqlQuery = @"insert into templates (url,
param_template,    method)    values    (@p_url,    @p_param_template,
@p_method)";

            await connection.ExecuteAsync(sqlQuery, parameters);
        }

        private async Task DeleteTemplateAsync(int templateId,
IDbConnection connection)
        {
```

```
                DynamicParameters parameters = new DynamicParameters();
                parameters.Add("p_templateId", templateId, DbType.Int32,
        direction: ParameterDirection.Input);

                string sqlQuery = @"delete from templates
                                    where id = @p_templateId";

                await connection.ExecuteAsync(sqlQuery, parameters);
            }

            private              async              Task<List<Template>>
        GetTemplatesAsync(IDbConnection connection)
            {
                string sqlQuery = @"SELECT id as Id, url as Url,
        param_template as ParamTemplate, method as Method FROM templates";

                List<Template>         templates         =         (await
        connection.QueryAsync<Template>(sqlQuery)).ToList();
                return templates;
            }

            public void Dispose()
            {

            }
        }
}
```

UsersRepository.cs

```
using System;
using Dapper;
using System.Data;
using System.Linq;
using Microsoft.Extensions.Logging;
using Bone.Data.Middleware;
using Bone.Domain.Repositories;
using Microsoft.Extensions.Configuration;
using Bone.Domain.Entities;
using System.Threading.Tasks;
using System.Collections.Generic;
```

```
namespace Bone.Data.Repositories
{
    public class UsersRepository : BaseRepository, IUsersRepository
    {
        public IConfiguration Configuration { get; }
        public UsersRepository(string connection, ILoggerFactory loggerFactory, IConfiguration configuration) : base(connection, loggerFactory)
        {
            Configuration = configuration;
        }

        public async Task<UserProfile> GetUserProfileAsync()
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }

                UserProfile userProfile = await GetUserProfileAsync(connection);
                return userProfile;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public async Task<UserProfile> PutUserProfileAsync(int id, UserProfile userProfile)
        {
            var connection = GetConnection();
            try
            {
                if (connection.State == ConnectionState.Closed)
                {
                    connection.Open();
                }
```

```
                await     UpdateUserProfileAsync(id,     userProfile,
connection);

                UserProfile        _userProfile        =        await
GetUserProfileAsync(connection);
                return _userProfile;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        private                async                Task<UserProfile>
GetUserProfileAsync(IDbConnection connection)
        {
            string sqlQuery = @"SELECT u.id as Id,
                                       u.user_name as UserName,
                                       u.email as Email,
                                       u.first_name as FirstName,
                                       u.last_name as LastName,
                                       u.phone_number           as
PhoneNumber,
                                       u.user_image as UserImage
                                  FROM user_profile u";

            UserProfile         userProfile         =         (await
connection.QueryAsync<UserProfile>(sqlQuery)).FirstOrDefault();
            return userProfile;
        }

        private async Task UpdateUserProfileAsync(int id, UserProfile
userProfile, IDbConnection connection)
        {
            string sqlQuery = @"update user_profile
                               set user_name = @p_user_name,
                                email = @p_email,
                                first_name = @p_first_name,
                                last_name = @p_last_name,
                                phone_number = @p_phone_number,
                                user_image = @p_user_image
```

```
                                    where id = @p_id";

                DynamicParameters parameters = new DynamicParameters();
                parameters.Add("p_user_name",       userProfile.Username,
        DbType.String, direction: ParameterDirection.Input);
                parameters.Add("p_email",              userProfile.Email,
        DbType.String, direction: ParameterDirection.Input);
                parameters.Add("p_first_name",     userProfile.FirstName,
        DbType.String, direction: ParameterDirection.Input);
                parameters.Add("p_last_name",       userProfile.LastName,
        DbType.String, direction: ParameterDirection.Input);
                parameters.Add("p_phone_number", userProfile.PhoneNumber,
        DbType.String, direction: ParameterDirection.Input);
                parameters.Add("p_user_image",     userProfile.UserImage,
        DbType.String, direction: ParameterDirection.Input);
                parameters.Add("p_id",   id,   DbType.Int16,   direction:
        ParameterDirection.Input);

                await connection.ExecuteAsync(sqlQuery, parameters);
            }

            public void Dispose()
            {

            }


        }
}
```